Information Sciences 645 (2023) 119190

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Building robust neural networks under adversarial machine learning attacks by using biologically-inspired neurons

Hossien Ali Ghiassirad [a], Faezeh Farivar [a,b], Mahdi Aliyari Shoorehdeli [c], Mohammad Sayad Haghighi [d,*]

[a] Department of Computer and Mechatronics Engineering, Science and Research Branch, Islamic Azad University, Tehran, Iran
[b] School of Computer Science, Institute for Research in Fundamental Sciences (IPM), P.o. Box 19395-5746, Tehran, Iran
[c] Department of Mechatronics Engineering, Faculty of Electrical Engineering, K. N. Toosi University of Technology, Tehran, Iran
[d] School of Electrical and Computer Engineering, University of Tehran, Iran

ARTICLE INFO

Keywords:
Deep neural network
Robustness
Adversarial machine learning
Adversarial examples
Binomial neuron
Cyber security
Artificial intelligence
Reliability

ABSTRACT

At the core of every artificial neural network, lie neurons which are approximate models of their biological counterparts. However, networks that are made of artificial neurons can become unreliable in the face of adversarial machine learning attacks. We traced this issue back to its roots and created a reliable artificial neuron (called binomial neuron) that can resist adversarial attacks such as DeepFool, Fast Gradient Sign Method (FGSM), Pixel, Square and Carlini and Wagner's (C&W) better. We propose to replace the input value of an artificial neuron with the expectation of a Bernoulli random variable. This random variable is made up of the spikes fired by the previous-layer binomial neuron upon crossing a randomly generated threshold. Depending on how long the neuron waits to aggregate the spikes and find the expectation, different model variants can be created. We conducted extensive experiments to evaluate the reliability of this new neuron. The results confirm the increase in the robustness/accuracy under adversarial attacks, or equivalently, the substantial increase in the amount of distortion attackers need to add to images in order to create successful adversarial samples. These distortions pass the perceptibility range to the extent that approach total image destruction.

1. Introduction

Visually imperceptible adversarial images are used by attackers to mislead trained artificial neural networks (ANN) in the context of adversarial machine learning [1]. Such threats are becoming more alarming, especially with the advent of Industry 4.0, big data, and the concept of deep learning as a service [2,3]. Imagine how dangerous it would be if we drove in autonomous cars whose Artificial Intelligence (AI) units could be deceived.

A reliable ANN should remain robust under attacks to the extent that the attacker has to significantly alter original inputs to make adversarial examples work [4]. In case of images, this change must be so big that the adversarial examples/samples become visually corrupted or destructed. The attackers may use optimization algorithms to reduce the alteration amount and increase the impact [5,6]. As a rule of thumb, the more visually corrupted the adversarial images get, the more robust and reliable the network

* Corresponding author.
E-mail addresses: sayad@ieee.org, sayad@ut.ac.ir (M. Sayad Haghighi).

https://doi.org/10.1016/j.ins.2023.119190
Received 23 July 2022; Received in revised form 14 May 2023; Accepted 16 May 2023

Available online 19 May 2023

becomes, especially under attacks. A research question can be asked here; How does the neural network of a human brain resist these attacks? Can the operation principles of biological neurons be an inspiration for building robust artificial neural networks?

A biological nervous system almost always works in the presence of randomness or noise [7]. Neurotransmitters are released stochastically at biological synapses [8]. Random signals have also been shown to exist in neural circuits [9]. Inspired by studying those noises in nature, this article introduces stochastic binomial neurons. It is shown that stochastic binomial neurons can be more robust against adversarial attacks than traditional neurons and work reliably under stronger attacks by effectively pushing the boundaries away from border samples and making decisions based on principal components.

Binomial neurons are a type of stochastic artificial neurons (e.g. Bernoulli neurons [10]) that produce and accumulate random binary values and exhibit several characteristics: 1. Biological plausibility, 2. Robustness against adversarial attacks, and 3. The ability to approximate full-precision artificial neurons over time. Furthermore, the possibility of being utilized as a regularizer (as defined in [11]) in the learning phase of full-precision ANNs is yet another advantage.

In this paper, we employ binomial neurons as a defence mechanism against adversarial attacks. This approach can be considered as a category of strategies that propose to modify the model itself in order to make it more reliable under attacks. Modifications may include changing the activation function or loss function of network layers. Binomial neurons have randomness at their outputs (i.e. activation functions), their inputs (i.e. membrane potentials) and overall output of the network, which make the network more robust against noise.

Biological neurons fire spikes whenever their membrane potential passes a threshold. These spikes occur asynchronously (unlike clock-driven synchronous ones), and so they propagate information via synapses under their strength (synaptic "weight"). The membrane potential of a biological neuron, which is called the post-synaptic potential (PSP), can be estimated based on a summation of the spikes' voltages from the pre-synaptic neurons, on the post-synaptic neuron. The duration of spikes is normally 1 to 2 ms, and the type of spikes, whether inhibitory or excitatory, is determined by the sign of the synaptic weights [12].

Our proposal is to introduce a type of stochastic spiking neuron whose behavior at time t is calculated by the firing rates of pre-synaptic neurons [10]. The stochastic spiking model of biological neurons [13] introduces a stochastic threshold to capture the randomness in their behavior. For the proposed stochastic neuron (whose spikes can be considered as the outputs of a Bernoulli random variable), the spiking probability is relevant to the current membrane potential of the neuron (e.g. via using the sigmoid function [14]).

Our idea focuses on the statistical properties of the spike population of neurons and that those properties (e.g. expected values) can carry information over time. We propose a model of stochastic spiking neurons that emphasizes the computational ability of the spike population rather than fidelity to biology.

The benefits of injecting randomness in the train and test phases are twofold: 1. The injection regularizes the inputs (over different runs) and creates robustness in training (similar to what data augmentation techniques do) 2. It simulates potential adversarial attacks during testing. The proposed neurons, from another viewpoint, adopt randomness as their intrinsic characteristic. The above two factors help the proposed neurons operate more reliably under adversarial attacks. Throughout the paper, stochastic spiking neural network (SSNN) and stochastic binary neural network are used interchangeably. The contributions of this paper can be summarized as follows:

- We propose a new type of neuron to increase the robustness of machine learning models against adversarial attacks. We interpret each input to an artificial neuron as the expectation of a binomial random variable. This means that the neuron's output is either 0 or 1, but a float output is constructed using an unbiased estimator on the binary outputs. The term "binomial neuron" is derived from the binomial distribution of the outputs. Similarly, we replace the error variable of an artificial neural network with an unbiased estimation of it (Section 3.2).
  The proposed binomial neurons are more biologically plausible than traditional full-precision neurons. Binomial neurons, similar to biological neurons, accumulate the evidences (in the form of binary spikes) before producing the output [15], and have some inherent randomness involved in the process (Section 5.1). This is different from the previous efforts in which neurons simply produced float numbers as outputs or those that artificially injected randomness into the network [16–19].
- We propose a method of using our binomial neurons for the purpose of regularization. One can train a network of binomial neurons and transfer its weights to a network of traditional full-precision neurons. We demonstrate that this typically increases the accuracy of the target network (Section 4.3).
- Through extensive and comparative experiments conduced on two datasets (CIFAR10 and MNIST), we demonstrate that a network of binomial neurons is more robust and reliable under adversarial machine learning attacks (such as DeepFool and FGSM) than the ones made of full-precision neurons. To attack a network of binomial neurons, the attacker has to distort the adversarial samples so much that the distortions are visually perceptible. The required distortion threshold is elevated to the extent that the attacker sometimes has to practically destroy the original sample to succeed. This implies higher robustness to adversarial attacks (Section 4.4).

The rest of the paper is organized as follows: In Section 2, related studies will be briefly reviewed. Section 3 introduces the proposed binomial neurons and explains how error is calculated in the networks that employ them. A training algorithm for such networks is introduced afterwards. In Section 4, some applications of binomial neural networks are given. We evaluate the reliability of such networks under adversarial attacks in this section. Then, in the next section, the results are discussed and, finally, in Section 6, the paper is concluded.

Table 1
Different types of injecting noise/randomness to defend against adversarial attacks.

Model	Phase	Randomness	Noise Position	Neuron output
Zhang et al. [17]	Test	Gaussian, discretization of pixels	input data	Float
Pinot et al. [18]	Train & Test	Exponential	input data	Float
He et al. [19]	Train	Gaussian	training parameters, activation function & input data	Float
Addepalli et al. [31]	Train & Test	Gaussian	feature maps	Float
Cohen et al. [32]	Train & Test	Gaussian	input data	Float
Eustratiadis et al. [30]	Train	Anisotropic noise	training parameters, activation function & input data	Float
Proposed Work	Train & Test	Binomial	Stochastic activation function	Binary

2. Related work

In the field of adversarial machine learning, evasive attack methods seek to make adversarial samples with minimal perturbations to the original test images that remain (almost) undetectable to humans perceptually [20,1,21,22] but cause models to misclassify those samples. Defense methods have different strategies for preventing attacks from misleading models. The categorization of these strategies is not the same in the literature [23–25], but all surveys and reviews list almost the same strategies. Approximately, all the defense strategies increase the amount of needed perturbation over the original image to make an adversarial sample, although some have tried to remove adversarial noise for this purpose [26]. Robustness is achieved if the required amount of perturbation leads to detection of anomalies by human eyes or machines.

To measure the amount of perturbation in an adversarial sample, structural similarity index (SSIM) [27] is popularly used. This index measures the amount of similarity between the original and adversarial images and can be used to show the degree to which adversarial images are recognizable to human eyes [23].

Injecting randomness, as a method of regularization or generalization (e.g. Dropout [28]), may make the network more robust against adversarial attacks [19], though there are some evidences showing that regularization does not always provide robustness [29].

There are a few works that have used randomness to make neural networks more robust. They can be categorized according to the injection phase, type of randomness (e.g. distribution of noise), and position of injection (refer to Table 1).

The idea of Parametric Noise Injection (PNI) [19] suggests adding some Gaussian noise to the input of network during training, which is similar to data augmentation. Moreover, in the PNI method, different Gaussian noises can be added to the activation function of each neuron and synaptic weights of the neural network. The mean and variance of such Gaussian noise(s) could be learned using the stochastic gradient descent (SGD) method during the training phase. Gaussian noise can also be added to the output of the feature extracting part of deep neural network [30]. Then, the classification layer(s) can be trained using a loss function called Weight-Covariance Alignment (WCA).

In contrast to the PNI method, another work suggests adding randomness not only during training, but also during testing time [18]. This work utilizes a family of exponential noises and also presents a theoretical justification on how adversarial attacks are thwarted through randomness. There is another injection of randomness during the test phase which uses clustering over pixel intensities of input data and constitutes another defence mechanism called Randomized Discretization (RandDisc). The first step of RandDisc method is adding a Gaussian noise to each pixel of the input image and then doing randomized discretization. Randomized smoothing is another technique that transforms the original classifier into a smoother one to include Gaussian random noise in input samples during the test phase [32]. Another work which also uses this technique implements such transformation in the feature space in order to include Gaussian noise [31]. In a more recent work, in addition to adding Gaussian noise to training data and synaptic weights as a defence method, an attack detection method, called Confidence Interval (CI), has been proposed, which helps to detect an adversarial example as an outlier [16].

This paper proposes binomial neurons that have randomness embedded at their core. Randomness injections are done based on Poisson binomial and binomial distributions. The injection happens during both training and testing, in activation functions and neuron inputs. The detailed model of this new neuron is presented in the next section. The network of binomial neurons will be applied to CIFAR10 and MNIST datasets under two scenarios; 1) an adversarial attack, and 2) a normal/benign condition which is devoid of adversarial attacks. We will present regularization and robustness properties of the proposed binomial neurons at the end.

3. The proposed model

In this section, we introduce the proposed model. Our model only concentrates on feed-forward networks (and not on the recurrent ones), and can be considered as a rate coding model rather than a temporal one.

3.1. Introduction of the new neuron

The traditional model of an artificial neuron can be described as follows:

$$u = b + \sum_{l=1}^{m} w_l . x_l$$
$$y = h(u) \tag{1}$$

where y is the output of the neuron, $h(.)$ is the activation function, b is the bias term, $w_l; l = 1, \dots, m$ are free parameters or synaptic weights of the neuron, $x_l; l = 1, \dots, m$ are the inputs to the neuron (that may be the outputs of the pre-synaptic neurons), and m is the number of inputs to the neuron.

This paper introduces the binomial neuron by suggesting to replace the outputs of the pre-synaptic neurons, i.e. x_l, in the traditional artificial neuron model of Eq. (1) with the expectation of a Bernoulli random variable like i_l. Note that $0 \le x_l \le 1$ (i.e. x_l as the output of a pre-synaptic neuron, should come from a squashing function like $max(0, tanh(.))$ or $sigmoid(.)$ or ...). To summarize, for the binomial neuron k, we have,

$$u_k = b_k + \sum_{l=1}^{m} w_{lk} . E\{i_l\} \tag{2}$$

i_l takes values from the set $\{0, 1\}$ and $E\{.\}$ denotes the expectation operation over time. Note that we have a spike from the pre-synaptic neuron l only when $i_l = 1$.

As the law of large number suggests, $E\{i_l\}$ can be estimated by using an unbiased estimator similar to Eq. (3). A special case of such estimation has previously been introduced in [33].

$$E\{i_l\} \approx \frac{1}{T} \sum_{t=-T+1}^{0} i_l(t) \tag{3}$$

where $t = 0$ is considered to be the current time. The key constant T, in Eq. (3), indicates the length of the evidence accumulation window, i.e., the number of i_l samples included in the expectation approximation window. T can be considered as the number of feed-forward passes (or in short, the number of passes) for neuron l. T should be determined before starting the train or test phase, and can be the same for all the network neurons. As T approaches infinity, $\frac{1}{T}\sum_{t=-T+1}^{0} i_l(t)$ converges to $E\{i_l\}$, though this hypothetical case would incur infinite delay. If we do not do the dividing by T at the right side of the equation, that part will represent a binomial random variable (or actually, a Poisson binomial random variable).

Considering unbiased estimators, the membrane potential of the new stochastic binary neuron model can be calculated as follows:

$$u_k(t) \approx b_k + \sum_{l} w_{lk} \left(\frac{1}{T} \sum_{t'=-T+t+1}^{t} i_l(t')\right) \tag{4}$$

In practice, a desired value for T will be used in feed-forward passes and the training phase. Note that larger T values lead to more precise estimations.

On the other hand, as the proposed neuron model is a stochastic binary one, the output of neuron k, $O_k(t)$, is calculated as follows when $i_l(t) = 0, 1$, w_{kl} is the synaptic weight and $u_k(t)$ is the membrane potential at time step t:

$$U(t) \sim Uniform(0, 1)(t)$$
$$O_k(t) = \begin{cases} 1 & h(u_k(t)) \ge U(t) \\ 0 & h(u_k(t)) < U(t) \end{cases} \tag{5}$$

Here, $U(t)$ represents a random threshold for neuron k at time t (which is sampled from a uniform distribution in order to be simple and consistent with the traditional model).

The expectation of $O_k(t)$ is given by:

$$\begin{aligned} E\{O_k(t)\} &= 0 \times P(O_k(t) = 0) + 1 \times P(O_k(t) = 1) \\ &= P(U(t) \le h(u_k(t))) = CDF(h(u_k(t))) \end{aligned} \tag{6}$$

where $CDF(.)$ is the cumulative distribution function of U(t) which is defined as $CDF(x) = max(0, min(x, 1))$. Eq. (6) shows the proposed stochastic binary neuron model where $E\{.\}$ denotes the expectation operation over time and $E\{O_k(t)\}$ is the expected value of the spike being emitted by neuron k at time step t.

In the model of spiking neurons, the values are continuous (i.e. $0 \le E\{i_l\} \le 1$) but the actual input $i_l(t)$ of a neuron is either 0 or 1. It means that $E\{i_l\} \approx \frac{1}{T}\sum_{t=-T+1}^{0} i_l(t)$ is an accumulation of last T input spikes of neuron k over time. When $T = 1$, $u_k(t)$ represents

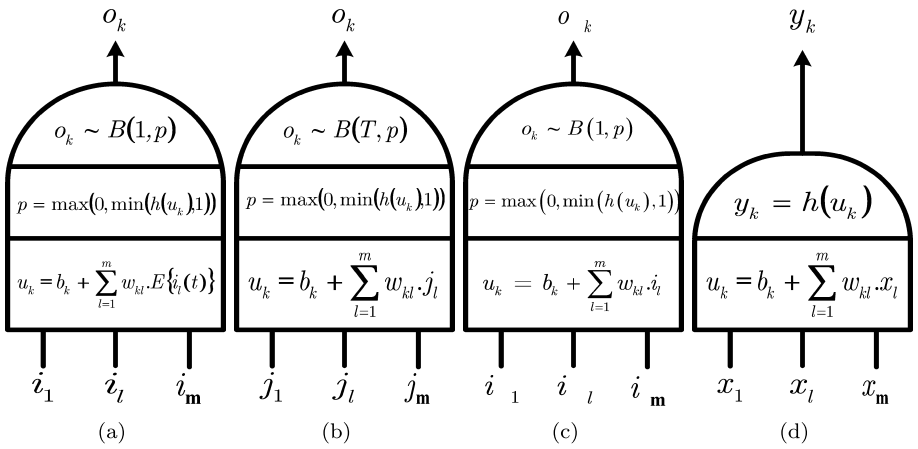

Fig. 1. Four types of neurons: (a) Type A, the proposed binomial neuron in which the input vector is $I = (i_1, \cdots, i_l, \cdots, i_m)$ where $i \in \{0,1\}$ and the output is $O \in \{0,1\}$. (b) Type B, the proposed artificial neuron with binomial activation function in which the input is $J = (j_1, \cdots, j_l, \cdots, j_m)$ where j is an integer that belongs to the interval $[0,T]$ and the output is $O \in [0,T]$. (c) Type C, an artificial neuron with Bernoulli activation function whose input I and output O are similar to those of neuron Type A [10]. (d) Type D, a traditional full-precision neuron in which $x \in \mathbb{R}$ and $y \in \mathbb{R}$.

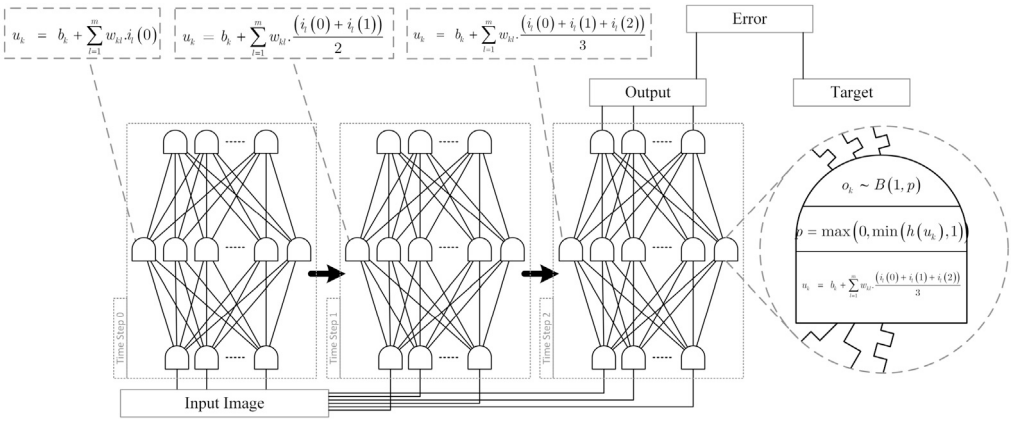

Fig. 2. Demonstration of the stages the input goes through to make the output along with the internals of neurons in a three-layer network when T = 3.

a weighted sum of Bernoulli random variables. Another type of such a weighted sum of Bernoulli random variables has previously been studied in [34].

Fig. 1 demonstrates different types of neurons that will be studied in this paper, i.e., binomial neurons (type A), neurons with binomial activation function (type B), Bernoulli neurons (type C) [10], and full precision neurons (type D). While binomial variables are used in both Type A and Type B neurons, what we refer to as a binomial neuron only encompasses Type A, and all the advantages listed in the contribution section are related to this type of neuron.

Fig. 2 demonstrates the steps taken to produce the output (and the corresponding error) from a three-layer network composed of binomial neurons. We have set T to 3 in this figure. At each stage of the process, the inputs to a neuron are stored and accumulated to compute the mathematical expectation. The right side of Fig. 2 shows the internals of the binomial neuron used in the network. It also shows that both inputs and outputs of the neurons are binary values.

3.2. Error in binomial neural networks

Using the proposed nonlinear model of Eq. (5), the expected value of a Bernoulli neuron output can be found as:

$$\begin{aligned} E\{O_k(t)\} &= P(O_k(t) = 1) \\ &= max(0, min(h(b_k + \sum_l w_{kl} E\{i_l(t)\}), 1)) \end{aligned} \tag{7}$$

Therefore, neuron k generates an output spike with a probability equal to $E\{O_k(t)\}$. The spike generation model of neuron k (Eq. (5)) ensures that the output of neuron k is a Bernoulli random variable. What neuron k should learn is the true value of its expectation while $Target(t)$ is the desired output (or ground truth) for the random variable of neuron k. Using the least square optimization approach, we define the error function as follows:

$$Error(t) = \frac{1}{2} E\{e(t)\}^2 \tag{8}$$

$e(t) = Target(t) - O_k(t)$ and $e(t)$ is a random variable itself. In practice, an unbiased estimation of $E\{e(t)\}$ will be used. In the training phase, the mean value of the error, which is also the mean value of the neuron mistakes, is:

$$\begin{aligned} E\{e(t)\} &= E\{Target(t) - O_k(t)\} \\ &= E\{Target(t)\} - E\{O_k(t)\} \end{aligned} \tag{9}$$

Bernoulli spiking is like tossing a coin in statistics [35]. It is one of the simplest and the most efficient methods of output generation in stochastic neurons [36].

3.3. The training algorithm

Learning algorithm determines how the elements of a model contribute to the overall task. One considerably successful learning method used in deep ANNs is the gradient descent algorithm [37]. The difficulty of employing gradient-based learning methods, such as back-propagation, for spiking neural networks lies in the calculation of the derivative of spike generator functions in neurons [38,39]. These derivatives are needed when the algorithm propagates the error to each neuron's weights. However, there are methods to solve this issue without resorting to gradient-based procedures [40]. There exist three solutions: 1. Using a differentiable pseudo-random number generator function as an activation function [41]; 2. Unbiased estimation for derivation of the activation function [41]; and 3. A straight-through estimation for the activation function [42]. This section explains how to train a network of binomial neurons using stochastic gradient descent and back-propagation.

3.3.1. Gradient-based learning algorithm for the proposed binomial neuron

A stochastic spiking neuron uses a pseudo-random number generating function ($U(t)$) to generate a value as the threshold which is used to decide whether neuron shall fire or not ($O_k(t) = 1$ means firing).

For the nonlinear neuron k, the probability of emitting a spike was given in Eq. (7). If $u_k(t)$ is not bounded by $h(.)$ in the interval $[0, 1]$, the value will be saturated. For example, if the nonlinear function is $tanh(.)$, a rectified version of $tanh$ like $max(0, tanh(.))$ will be used. Hence the value of $h'(.)$ (the first derivative of $h(.)$) at the back-propagation phase when $h(.) > 1$ *or* $h(.) < 0$ will be zero.

Using the expectation of $e(t)$ in the error function (Section 3.2) makes it possible for the probabilistic activation functions to be differentiable. Therefore, the gradient descent algorithm can be written as follows:

$$\begin{aligned} W_n &= W_{n-1} - \gamma \times \frac{\partial Error(t)}{\partial W_{n-1}} \\ &= W_{n-1} - \gamma \times \frac{\partial Error(t)}{\partial E\{e(t)\}} \times \frac{\partial E\{e(t)\}}{\partial E\{O_k(t)\}} \times \frac{\partial E\{O_k(t)\}}{\partial u_k(t)} \times \frac{\partial u_k(t)}{\partial W_{n-1}} \end{aligned} \tag{10}$$

Thus:

$$W_n = W_{n-1} - \gamma \times E\{e(t)\} \times (-1) \times h'(u_k(t)) \times E\{I(t)\} \tag{11}$$

where γ is the learning rate and $E\{I(t)\}$ is the vector form of $E\{i_l(t)\}$. Equation (11) is the final equation needed from the gradient descent method to train the binomial neurons. In practice, all the expectations in Eq. (11) will be replaced by their unbiased estimations, as in Eq. (3). Regarding $T = 1$ and utilizing Eq. (9), if estimation of the mathematical expectations (in Eq. (11)) is done during post error propagation ($\partial O / \partial h = 1$), the well-known technique of straight-through is used.

Algorithm 1 demonstrates a customized version of the stochastic gradient descent method to train an N-layer network of binomial neurons. The network structure is not affected by the replacement of binomial neurons. The protection capability against adversarial attacks comes along with the neurons themselves.

Part 1.1 of the algorithm will repeat in the test phase, exactly as it is here. This means that the neurons generate outputs stochastically in the test phase too.

In this algorithm, T, which can be any integer greater than 0, is the number of times that the network passes the inference process and accumulates the inputs to the neurons. These accumulated values (over time) are used to estimate the mathematical expectations in the neurons. When $T = 1$, a network of Bernoulli neurons will be formed, and when $T > 1$, the values created over time can form a binomial variable (or by approximation, a Poisson variable). The reason for this approximation is the changes in the outputs of the neurons in each inference of the network. The greater the value of T is, the more similar the internal computations of neurons will be to those of full precision neurons. It is worth mentioning that this algorithm is meant to implement the proposed method on synchronous processors.

To demonstrate the impact of time on the mathematical expectation values, a comparison between the results of training a network of binomial neurons (i.e. Type A) using Algorithm 1 and other types of mentioned neurons (i.e. Type B, C, and D) is done in the evaluation section. The training is carried out using the SGD method. According to Fig. 1, one of these neuron types is Type B whose output generates a binomial random variable instantly. The output of this type of neuron is not just 0 or 1. It depends on T and takes values between 0 and T $(0, 1, 2, \ldots, T)$.

3.3.2. Complexity analysis

There are four computational characteristics in binomial neurons that differentiate them from traditional neurons:

Algorithm 1 Training an N-layer network made of the proposed neurons with $K^{(n)}$ neurons in the nth layer.

Require: a minibatch of inputs and targets (I^0 , $Target$), previous weights W , and learning rate γ.
Ensure: updated weights W'

1. Computing the parameters gradients:
1.1 Forward propagation:
for $t \leftarrow 1, T$ **do** ▷ T forward passes
  for $n \leftarrow 1, N$ **do** ▷ Traverse n layers
    for $l \leftarrow 1, K^{(n-1)}$ **do** ▷ $E\{O^{(n-1)}(t)\}$ over t
      $i_l^{(n)} \leftarrow \frac{1}{t} \sum_{z=1}^{t} O_l^{(n-1)}(z)$
    for $k \leftarrow 1, K^{(n)}$ **do** ▷ For all neurons of layer n
      $u_k^{(n)}(t) \leftarrow b_k + \sum_l w_{kl} \times i_l^{(n)}$
      $U \leftarrow Uniform(0, 1)$ ▷ Random threshold
      if $h(u_k(t)) > U$ **then** ▷ Bernoulli Activation
        $O_k^{(n)}(t) \leftarrow 1$
      else
        $O_k^{(n)}(t) \leftarrow 0$
  if $t = T$ **then** ▷ Calculate $Error$ according to Eq. (9)
    $EO \leftarrow E\{O^{(N)}(t)\}$
    $SO \leftarrow Softmax(EO)$
    $Error \leftarrow Loss(SO, Target)$
1.2 Backward propagation:
for $n \leftarrow N, 1$ **do**
  $\delta \leftarrow \frac{\partial Error}{\partial E\{O^{(n)}(t)\}}$
  for $k \leftarrow 1, K^{(n)}$ **do** ▷ For all neurons of layer n
    $\Delta W_k^{(n)} \leftarrow \delta \times \frac{\partial h(u_k(t))}{\partial W_k^{(n)}}$
2. Accumulating the parameters gradients:
for $n \leftarrow N, 1$ **do**
  for $k \leftarrow 1, K^{(n)}$ **do** ▷ For all neurons of layer n
    $W'^{(n)}_k \leftarrow W_k^{(n)} - \gamma \times \Delta W_k^{(n)}$

1. The pseudo-random Bernoulli activator function,
2. The mathematical expectations,
3. The potential to replace dot product between layers with bitwise 'AND' operator,
4. T constant that determines the number of feed-forwards.

If K is the number of neurons in each layer and we assume a constant overhead due to the Bernoulli activation function in binomial neurons (item 1), the complexity of calculating the output of binomial neurons (in each layer) is of $O(K)$. When the number of neurons for two consecutive layers is K, the inter-layer calculations (i.e. the dot products) are of $O(K^2)$. At the input of each binomial neuron (for item 2), an 'ADD' and a 'DIV' operator are needed to make the mathematical expectation. The expectation is practically carried out over multiple runs (which contribute to the expectation). Therefore, in each run, 'DIV' can be embedded into the weights during implementation without the need for additional calculations. The addition is still required in each run though. However, since this expectation operation will be of $O(K)$ in each layer, the overall layer-based computational complexity will remain at $O(K^2)$.

Based on item 3, the dot product between layers can be simplified to bitwise AND operation, however, despite the simplification, the inter-layer calculations in a network of binomial neurons will still be of $O(K^2)$. Regarding T (item 4), if the complexity of a feed-forward calculation for a network of traditional neurons with N layers (and K neurons in each layer) is of $O(N \times K)$, the complexity of a similar Binomial network is of $O(T \times N \times K)$. This is the computational complexity for inferring an input sample during the training and the testing phases. There is no difference (in complexity) between the binomial and traditional networks in error propagation and weight update of the SGD algorithm.

4. Evaluation results

We design three experiments:

1. Testing the performance of binomial neurons in classification scenarios on MNIST and CIFAR-10 datasets.
2. Evaluating the robustness/reliability of networks composed of binomial neurons against adversarial attacks by creating adversarial samples and measuring the degree of visual distortions.
3. Finding the best experimental configuration.

We first explain the setup and architecture of the test and then present the results.

4.1. Benchmarks

MNIST is a dataset of handwritten digits (0–9) [43] and is used as one of the famous performance benchmarks in the field of machine vision. This dataset includes 60,000 $28 \times 28 px$ gray images for training and 10,000 images for testing.

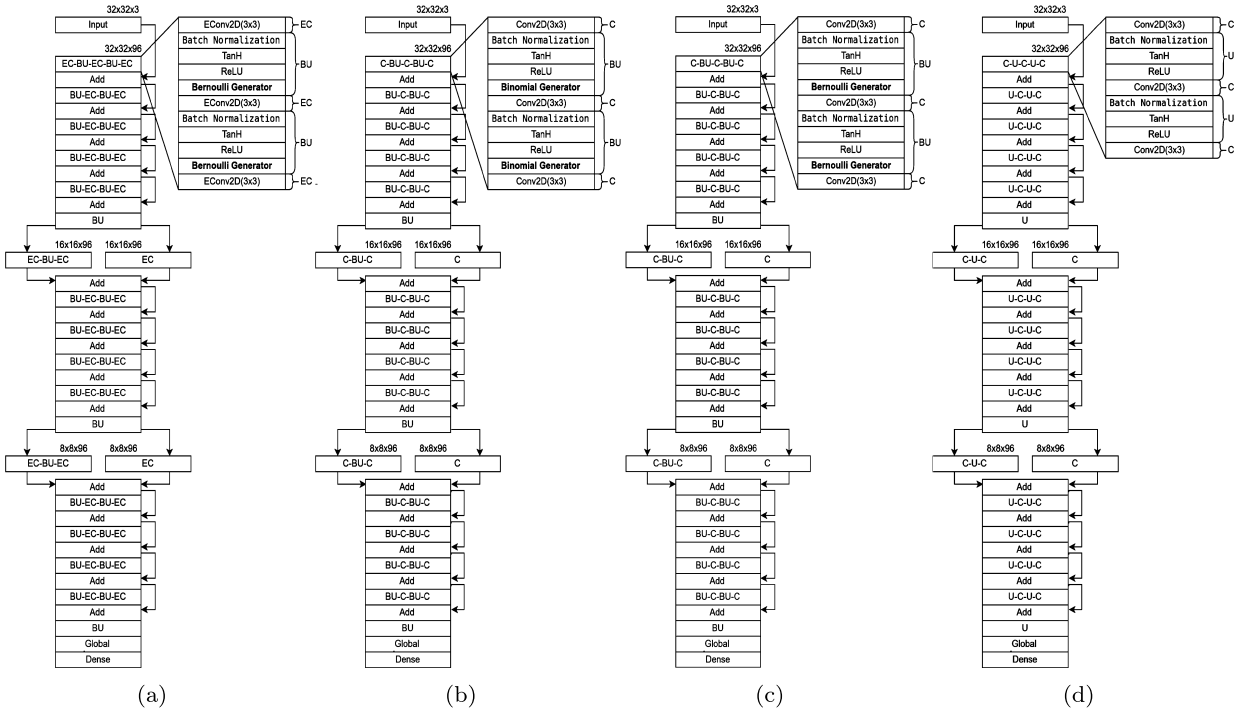

Fig. 3. The architectures used in our experiments. (a) The network used to check the performance of Type A neurons. (b) The network used for Type B neurons. (c) The network used for Type C neurons. (d) The network used for Type D neurons.

CIFAR-10 [44] is a labeled subset of 80 million images and one of the challenging classification problems in the field of machine vision. The set consists of 60,000 $32 \times 32px$ color images in 10 classes, with 6,000 images per class out of which 5000 are meant to be used for training and 1000 for testing purposes.

4.2. The image classification architecture

The classification task is a type of function approximation that maps samples of a cause to a class. Each class is associated with an output neuron. For n classes, the neural network usually has n output neurons. The number of input neurons of the network indicates the dimension of the input images. The input value $px_k(x, y)$ of the pixel (x, y) of image k is a real number in the interval [0,1].

Another important point is the times a sample should be presented to a network in the training or the inference phase. In some machine learning models such as ANN, one data sample is fed to the network in each iteration (i.e. one step calculation in the whole network). In our model, the learning algorithm needs a delay (equal to the number of iterations, that is T) before it can feed a data sample to the network.

The architectures of the networks used for MNIST and CIFAR-10 are depicted in Fig. 3. The EConv2D layers accumulate input spikes (0 s or 1 s) over time. For MNIST, the architectures have lower number of maps, which means the output maps in the first part of each network have a dimension of $28 \times 28 \times 16$, in the second part have a dimension of $14 \times 14 \times 32$ and in the third have a dimension of $7 \times 7 \times 64$.

According to Eq. (6), the Binomial Unit (BU) is modeled through using $Tanh(.)$ function, which constrains the neuron membrane potential $u_k(t)$ to the values between -1 and 1. The output of $h(.)$, and then $Tanh(.)$ pass through the $ReLU(.)$ or $max(0, .)$ to bring the result between 0 and 1. A spike is then generated through stochastic thresholding. For this purpose, a new activation function (BU) is implemented. The Dense layer is composed of perceptrons with $softmax$ activation function and $l2$ regularizers, preceded by a Global Average Pooling 2D layer.

The TensorFlow [45] library was used to train the networks on a GPU (Geforce 2080Ti). For the expected values, the spikes generated by the BU functions were accumulated over time only for the last T spikes at the output *blob* of the BU layer. According to Eq. (3), the expectation of the input, i.e. $E\{i_l\}$, is presented to each layer after assessing the output *blob* of the previous BU layer.

The built-in uniform pseudo-random generating function of TensorFlow was applied to generate the threshold $U(t)$ (in Eq. (5)). Random number generation was asynchronous across all neurons but they had the same speed at which they ran a fixed number of processing cycles within the T time steps. Other TensorFlow hyper-parameters are reported in Table 2.

Table 2
TensorFlow hyper-parameters used in the experiments.

Parameter	Value
kernel initializer	he_normal
optimizer	SGD (lr = .1, momentum = 0.9, nesterov = True)
loss	categorical_crossentropy
batch size	32
max epoch	250
targets	categorical (binary class matrices)

Table 3
Results for the MNIST dataset.

Row	Type in Train	Train T	Type in Test	Test T	Accuracy (%)
1	B	2	D	∞	99.73
2	B	2	B	5	99.67
3	B	3	D	∞	99.64
4	B	2	B	3	99.63
5	A	2	D	∞	99.61
6	A	3	D	∞	99.61
7	C	1	D	∞	99.6
8	D	∞	D	∞	99.59
9	B	2	B	2	99.57
11	B	3	B	5	99.56
10	C	1	B	5	99.56
12	C	1	B	3	99.55
13	C	1	B	2	99.53
14	A	3	C	1	99.52
15	A	3	A	3	99.51
16	B	3	B	3	99.51
17	A	3	A	2	99.5
18	A	3	A	5	99.5
19	A	2	C	1	99.49
20	A	2	A	2	99.49
21	A	2	A	3	99.49
22	A	2	A	5	99.47
26	B	3	B	2	99.45
23	C	1	A	2	99.45
24	C	1	A	3	99.45
25	C	1	A	5	99.45
27	C	1	C	1	99.44
28	C	1	C	1	99.44
29	D	∞	B	5	99.43
30	B	2	C	1	99.38
31	B	3	C	1	99.23
32	D	∞	B	3	99.22
33	D	∞	B	2	98.78
34	D	∞	C	1	91.82
35	D	∞	A	2	82.92
36	D	∞	C	1	82.91
37	D	∞	A	5	82.91
38	D	∞	A	3	82.9

4.3. Image classification using binomial neurons

In order to compare the performance of the proposed neuron with that of other types (i.e. the four types of neurons shown in Fig. 1), we conducted different experiments on two popular datasets, namely MNIST and CIFAR10. The detailed results are reported in Table 3 and Table 4. These tables present the results of averaging 10 executions in each experiment, which means that the inference process has been performed 750 times, and through averaging, 75 numbers are presented in the two tables.

Regarding the uncertainty in computations in the neurons, since the results of classifying the data during each process of inference can vary, the results are produced by averaging 30 executions (in each experiment). It is worth noting that in all the experiments, no data augmentation was used on the input data to better showcase the impact of regularization.

According to Table 3 and Table 4, some findings from studying MNIST and CIFAR10 datasets are:

Table 4
Results for the CIFAR-10 dataset.

Row	Type in Train	Train T	Type in Test	Test T	Accuracy (%)
1	A	3	D	∞	92.17
2	A	2	D	∞	91.96
3	C	1	D	∞	91.29
4	B	2	D	∞	91.16
5	C	1	B	5	90.93
6	B	3	D	∞	90.89
7	C	1	B	3	90.69
8	A	3	C	1	90.62
9	B	2	B	5	90.62
10	A	3	A	5	90.53
11	A	3	A	2	90.47
12	A	3	A	3	90.44
13	C	1	B	2	90.42
14	A	2	A	3	90.25
15	A	2	A	5	90.22
16	A	2	A	2	90.2
17	A	2	C	1	90.19
18	B	2	B	3	90.17
19	B	3	B	5	90.12
20	B	3	B	3	89.65
21	B	2	B	2	89.52
22	C	1	C	1	89.41
23	C	1	A	3	89.3
24	C	1	C	1	89.27
25	C	1	A	2	89.22
26	C	1	A	5	89.19
27	B	3	B	2	88.88
28	B	2	C	1	87.28
29	B	3	C	1	86.02
30	D	∞	B	5	83.56
31	D	∞	B	3	80.78
32	D	∞	B	2	76.69
33	D	∞	D	∞	73.36
34	D	∞	C	1	61.28
35	D	∞	C	1	32.7
36	D	∞	A	2	32.69
37	D	∞	A	5	32.54
38	D	∞	A	3	32.48

- Training under different bounded T values and testing with $T = \infty$ (Type D neuron) regularized the network (for both MNIST and CIFAR10). Additionally, the best result was achieved when testing was done with $T = \infty$ for every type, which is due to using the exact values of mathematical expectations at the outputs.
- It is possible that using different T values in training and testing result in higher accuracies, compared to using the same T in both. For example, for $T = 1$ in the test phase, the accuracy of classification with $T = 2$ (Row 17 for CIFAR10 and Row 19 for MNIST) is higher than that of $T = 1$ (Row 22 for CIFAR10 and Row 27 for MNIST) in the training. A p-value of lower than 0.003 (in 2-sample paired t-test) for both datasets shows how significant the change in the accuracy is.
- The experiments do not provide enough evidence to confirm that changing T in the test phase (from the same value in the training phase) can improve accuracy in type A neurons. More precisely, their p-values are larger than 0.05. This could originate from the uncertain (stochastic) behavior due to changes in the approximate expectation values calculated during each inference of the network.
- The experiments show that changing T in the test phase (with respect to the T used in the training) can improve accuracy in Type B neurons, while the p-value of 2-sample paired t-test of all comparisons is 0 for a 3-digit precision.

4.4. Attacking networks of binomial neurons

In this paper, three white-box adversarial machine learning attacks, namely, Carlini and Wagner's (C&W) [46], DeepFool [20] and FGSM [1], as well as two black-box ones, i.e. Square [21] and Pixel [47] attacks, have been applied on networks made of different types of neurons. Fig. 4 has listed ten configurations with the highest accuracies. It also shows the performance of traditional neurons as the baseline. The impacts of these attacks on the test sets are different depending on the neuron type, T values in the train/test phases, and attack methods. Notice that accuracy alone is not a figure of merit in adversarial machine learning. The amount of added perturbation is also important. The perturbations of test images in the two datasets for the networks made of the four different neuron types are depicted in Fig. 5. To quantify the amount of distortions, the SSIM [27] metric has been utilized (see Fig. 4). This

Row	Type in Train	T in Train	Type in Test	T in Test	Accuracy	DeepFool Accuracy	FGM Accuracy	Square Accuracy	C&W Accuracy	Pixel Accuracy	DeepFool SSIM	FGM SSIM	Square SSIM	C&W SSIM	Pixel SSIM
baseline	D	∞	D	∞	73.29	13.14	34.89	5.19	70.73	70.5	62.66	91.7	73.76	95.55	94.08
1	A	2	D	∞	92.14	4.64	33.98	5.08	81.11	90.08	22.15	92.49	67.52	94.3	94.45
2	A	3	D	∞	92.05	4.19	34.81	4.61	80.35	89.98	21.26	92.45	67.64	94.53	94.27
3	C	1	D	∞	91.22	5.12	27.31	3.27	90.9	88.53	24.21	92.41	68.02	95.77	96.74
4	C	1	D	∞	91.17	4.98	32.89	5.45	84.25	87.85	27.66	92.49	67.99	95.77	95.06
5	B	2	D	∞	91.16	4.91	27.75	5.07	89.7	87.8	23.53	92.47	67.44	94.69	96.9
6	C	1	B	5	90.99	7.44	28.1	26.38	90.99	88.46	25.35	92.41	67.58	99.92	97.03
7	B	3	D	∞	90.89	5.04	26.33	4.84	91.2	87.16	23.91	92.49	67.78	95.34	97.05
8	C	1	B	3	90.7	8.23	28.41	30.21	91.05	88.27	25.84	92.41	67.4	99.91	97.09
9	B	2	B	5	90.63	7.07	28.83	35.39	90.06	87.7	25.34	92.47	67.53	99.96	97.23
10	A	3	C	1	90.5	10.22	37.13	41.65	89.28	88.2	28.25	92.46	66.8	99.81	94.75

(a)

Row	Type in Train	T in Train	Type in Test	T in Test	Accuracy	DeepFool Accuracy	FGM Accuracy	Square Accuracy	C&W Accuracy	Pixel Accuracy	DeepFool SSIM	FGM SSIM	Square SSIM	C&W SSIM	Pixel SSIM
baseline	D	∞	D	∞	99.54	8.43	94.46	11.36	98.37	99.58	3.94	72.14	51.8	99.99	99.52
1	B	2	D	∞	99.73	4.53	97.52	0.44	98.3	99.67	3.52	71.06	52.3	99.98	99.81
2	B	2	B	5	99.66	3.92	97.36	18.17	99.45	99.61	3.66	71.04	51.86	100	99.82
3	B	3	D	∞	99.64	4.47	96.85	5.87	96.6	99.57	3.57	71.09	51.71	99.96	99.7
4	C	1	D	∞	99.62	5.79	97.47	2.22	99.22	99.54	4.84	71.13	52.58	99.98	99.85
5	A	2	D	∞	99.62	7.43	97.72	2.52	97.27	99.5	3.39	71.05	51.67	100	99.41
6	B	2	B	3	99.61	4.32	97.08	19.82	99.4	99.56	3.85	71.03	51.72	100	99.82
7	D	∞	D	∞	99.6	5.52	90.8	5.63	98.4	99.5	3.43	71.33	52.92	99.83	99.74
8	A	3	D	∞	99.59	6.99	97.73	3.28	98.9	99.5	3.79	71.08	51.53	99.99	99.57
9	C	1	D	∞	99.58	5.43	96.95	2.24	98	99.5	3.97	71.13	52.6	99.98	99.76
10	B	2	B	2	99.58	4.58	96.7	23.85	99.27	99.49	4.14	71.03	52.52	100	99.8

(b)

Fig. 4. (a) CIFAR10, Best accuracies with different T values in train and test, (b) MNIST, Best accuracies with different T values in train and test.

Fig. 5. DeepFool attack results on CIFAR10 for an airplane image sample. Title of each image indicates the value of T used for training (T_{tr}) and testing (T_{te}), followed by its SSIM score in the form of T_{tr},T_{te} = SSIM. The images show the attack cannot make adversarial samples with low perturbation levels when binomial neurons are employed (apparently this is when $T_{tr} > 1$).

metric helps to understand the extent to which an attack has to distort an image sample to achieve a certain level of accuracy (or inaccuracy). If there is no limit on the amount of perturbation, any sample can be transformed into any other.

4.5. Making the most reliable binomial neural network

Alpha score is defined to help in finding the best experiment configuration for robustness and reliability.

$$\alpha \triangleq a \sum_i \frac{f_i}{s_i} \tag{12}$$

Here, a is the accuracy of network when no attack is present, f_i is the accuracy of network when attack i is present, and s_i is the SSIM score related to attack i. A good network is one with high alpha score, i.e. having high accuracy both under attack and in normal conditions. However, when it is under attack and the accuracy decreases, its SSIM score should also decrease significantly to enable visual perception and detection of adversarial perturbations. Equation (12) is more an optimistic metric for the performance

Row	Type in Train	T in Train	Type in Test	T in Test	Alpha Score	Accuracy	DeepFool Accuracy	FGM Accuracy	Square Accuracy	C&W Accuracy	Pixel Accuracy	DeepFool SSIM	FGM SSIM	Square SSIM	C&W SSIM	Pixel SSIM
baseline	D	∞	D	∞	157.584	73.29	13.14	34.89	5.19	70.73	70.5	62.66	91.7	73.76	95.55	94.08
1	A	3	A	5	295.328	90.39	10.52	36.65	43.62	90.84	88.3	28.09	92.46	66.43	100	94.81
2	A	2	A	2	294.993	90.15	10.72	36.13	45.38	90.39	88.14	28.76	92.5	66.92	100	95.13
3	A	2	A	5	293.754	90.35	10.87	35.67	44.17	90.26	88.26	29.04	92.49	66.79	100	95.17
4	A	3	A	3	293.392	90.39	10.32	36.36	43.25	90.87	88.23	27.96	92.46	67.21	100	94.74
5	A	3	A	2	293.024	90.45	10.07	36.83	42.75	91.04	88.2	27.63	92.46	67.25	100	94.8
6	A	2	A	3	292.88	90.23	10.71	35.93	44.22	90.6	88.21	29.5	92.5	66.84	100	95.17
7	A	2	C	1	292.844	90.3	10.83	36.01	43.73	89.89	88.19	28.81	92.5	67.14	99.93	95.14
8	A	3	C	1	290.706	90.5	10.22	37.13	41.65	89.28	88.2	28.25	92.46	66.8	99.81	94.75
9	C	1	A	3	288.374	89.14	11.44	36.5	47.46	89.42	86.72	34.95	92.49	66.65	100	95.63
10	C	1	A	5	287.559	89.33	11.54	36.59	46.59	89.24	86.71	35.05	92.49	67	100	95.66

(a)

Row	Type in Train	T in Train	Type in Test	T in Test	Alpha Score	Accuracy	DeepFool Accuracy	FGM Accuracy	Square Accuracy	C&W Accuracy	Pixel Accuracy	DeepFool SSIM	FGM SSIM	Square SSIM	C&W SSIM	Pixel SSIM
baseline	D	∞	D	∞	562.67	99.54	8.43	94.46	11.36	98.37	99.58	3.94	72.14	51.8	99.99	99.52
1	A	3	A	3	579.321	99.49	6.55	97.11	39.35	99.28	99.41	3.84	71.03	51.83	100	99.61
2	A	3	A	5	567.803	99.5	6.49	97.21	37.87	99.27	99.4	4.02	71.02	51.68	100	99.61
3	A	3	C	1	567.226	99.54	6.55	97.18	36.15	99.25	99.44	3.97	71.02	52.44	100	99.6
4	A	3	A	2	563.538	99.52	6.64	97.28	32.1	99.38	99.41	3.96	71.02	51.43	100	99.6
5	D	∞	D	∞	562.67	99.54	8.43	94.46	11.36	98.37	99.58	3.94	72.14	51.8	99.99	99.52
6	C	1	A	2	561.38	99.46	7.53	96.23	48.74	99.28	99.42	5.45	71.1	52.95	100	99.83
7	A	2	A	2	560.028	99.48	7.36	97	23.68	99.15	99.37	4.05	71.01	51.99	100	99.44
8	A	2	C	1	559.289	99.52	7.27	96.99	22.7	99.14	99.36	3.97	71.02	52.49	100	99.45
9	C	1	A	5	557.23	99.43	7.6	96.2	48.37	99.36	99.4	5.7	71.09	52.1	100	99.83
10	A	2	D	∞	556.825	99.62	7.43	97.72	2.52	97.27	99.5	3.39	71.05	51.67	100	99.41

(b)

Fig. 6. (a) CIFAR10, Top 10 alpha values for different T parameters in training and testing. (b) MNIST, Top 10 alpha values for different T parameters in training and testing.

under attacks as it uses summation of the reliability/robustness scores (i.e. $\frac{f_i}{s_i}$ values). A pessimistic version of the metric could use multiplication instead of summation.

Alpha score, as defined by Equation (12), indicates how effective a set of attacks are on a network. To analyze the security level of binomial neurons (in a comparison to traditional neurons), one can use this score as a metric. Fig. 6 shows the top 10 experiments with the highest scores along with their configurations. As it can be seen, the dominance of binomial neurons is rather obvious.

The depicted results show that binomial neural networks are more robust against adversarial machine learning attacks compared to those made of full-precision neurons. As a result, attackers need to add more perturbations and create more powerful and visually perceptible noises in the adversarial samples to mislead the network. A Binomial neural network is intrinsically trained in presence of randomness (with almost binomial distribution). Given that most attacks add adversarial randomness to input samples, probabilistically speaking, they are treated as the randomness seen by the network during training and can be prevented from causing destructive outcomes.

5. Discussions and findings

Although the proposed models have many benefits, they also come with some drawbacks. For example, achieving more accurate results takes more time (since greater number of feed-forward passes is required for the sake of inference). Additionally, accumulating binary values at the input of binomial neurons requires more hardware implementation compared to traditional neural networks, where neurons only copy their output values to the next round of neuron inputs. To further understand the dynamic aspects of the proposed idea, it is necessary to discuss some important subjects.

5.1. Biological plausibility of the proposed model

Considering Eq. (3), a binomial random variable is formed at the input of any binomial neuron of the proposed kind, which somehow simulates the accumulation of evidences in biological neurons over time. Binomial random variables converge (in distribution) to Poisson distribution over time (the law of rare events). The existence of Poisson distribution in spike populations of biological neurons has a strong basis [48,49]. From another perspective, these neurons use Poisson random variables during inference. However, analyzing this issue in depth is outside the scope of this article. Moreover, when $T \gg 0$, binomial variables can be approximated by a Gaussian function. This can model the Gaussian noise in the output of neurons. The presence of such noise in the output of neurons has been studied previously [50].

5.2. Comparing the model with an almost unanimously accepted stochastic firing neuron model

The essential quantity to be transmitted from one group of neurons to the next is the firing rate, which is defined either as a temporal or a population average. If this is true, models formulated on the level of firing rate will be sufficient [10].

Our model uses the population average method. In general, stochastic rate models average out an ensemble. These models consider the spike train of an arbitrary neuron as a stochastic process, and the probability of finding a spike in this train at t as the firing rate.

According to [10,13], for discrete time models, the underlying stochastic process includes the samples with active value ($O_k(t) = +1$) or quiescent value ($O_k(t) = 0$). Hence the probability of a neuron being active, given input u_k at time t, is:

$$P(O_k(t) = +1 | u_k(t)) = h(u_k) \tag{13}$$

where $h(.)$ is the gain function, and $u_k(t) = \sum_l w_{kl} O_l(t)$. Thus, the probability of $O_k(t)$ being active is not dependent on the previous states, but rather on the current state of pre-synaptic neurons. According to Eq. (2) and Eq. (3), the probability that the output $O_k(t)$ is active depends on the last T states of pre-synaptic neurons. Basically, what really happens in our model is the accumulation of spikes over last T time steps.

5.3. Characteristics of binomial neural networks

Using the proposed binomial neurons has both advantages and disadvantages. We list them here starting with the advantages:

- Since the model approximates artificial neuron outputs from binary values over time, it can be used to solve actual machine learning problems, similar to traditional neurons.
- The proposed way of estimating the error enables us to use the classic back propagation algorithm for the stochastic binary values in a network of binomial neurons.
- Stochastic firing of neurons makes the network more robust to noises and adversarial attacks.
- Binary computations in the network can be done with binary operands that lead to simpler hardware implementations and faster computations.
- Unlike traditional neurons that can only be used in synchronous networks, the proposed binomial neurons have the ability to also do a feed-forward computation in an asynchronous way. This type of implementation is useful in neuromorphic computing, and biological plausibility of binomial neurons further helps them excel in those applications as well.
- Binomial neurons, due to their inherent randomness originated from the virtual Bernoulli trials, create a non-deterministic neural network after deployment. This makes the network behavior unpredictable for attackers. Therefore, producing successful adversarial samples becomes harder and requires injecting more disturbance. This translates to more robustness from the network owner's perspective.

As a general rule, with advantages, come disadvantages. Some disadvantages of the proposed model are listed below:

- More accurate result needs more time for inference (more number of feed-forward passes). This is consistent with nature experiences.
- Accumulation of binary values in neurons might require additional memory cells in hardware implementations.
- Probabilistic outputs by binomial neurons may, to some extent, increase the computational overhead.

Despite all these, we believe the advantages well outweigh the disadvantages.

5.4. Research findings

In this part, we summarize the most important findings of this research. Some of these findings have been referred to in the results or discussion sections indirectly.

- We found how a neuron similar to the stochastic spiking neurons can estimate a full-precision neuron over time. We named it the binomial neuron.
- We found that mimicking biological neural characteristics can help to build robust neural networks. Due to the stochastic binary output, the proposed neurons can be used as a defense method against adversarial machine learning attacks.
- We found that using the weights of a pre-trained binomial network in a traditional full-precision network usually yields higher accuracies. Therefore, binomial neurons can be used as regularizers to decrease overfitting.

6. Conclusion

This article presented a novel model of stochastically spiking neurons inspire from biological neurons. The proposed neuron model accumulates binary values, or spikes, over time, and is capable of mimicking full precision neurons as a special case when given a large interval for spike aggregation. This capability can be considered as an advantage for stochastic binary neurons over deterministic binary neurons. A potential advantage of stochastic binary neurons is their ability to resist adversarial attacks. It is due to the fact that the proposed neurons are trained under randomness. Finally, in the inference phase, since the model bears a closer resemblance to brain than current ANNs (in terms of the computing strategy), the proposed model can potentially be used as a defense mechanism in adversarial machine learning.

CRediT authorship contribution statement

Hossien Ali Ghiassirad: Conceptualization, Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Faezeh Farivar:** Methodology, Supervision, Validation, Writing – original draft. **Mahdi Aliyari Shoorehdeli:** Methodology, Project administration, Supervision, Writing – original draft. **Mohammad Sayad Haghighi:** Conceptualization, Investigation, Methodology, Writing – original draft, Writing – review & editing.

Declaration of competing interest

The authors declare no financial interests/personal relationships which may be considered as potential competing interests.

Data availability

No data was used for the research described in the article.

References

[1] I.J. Goodfellow, J. Shlens, C. Szegedy, Explaining and harnessing adversarial examples, in: The 3rd International Conference on Learning Representations, 2015, pp. 1–11, arXiv:1412.6572.

[2] M.S. Haghighi, F. Farivar, A machine learning-based approach to build zero false-positive ipss for industrial iot and cps with a case study on power grids security, preprint, arXiv:2004.06432.

[3] M.S. Haghighi, A. Sheikhjafari, A. Jolfaei, F. Farivar, S. Ahmadzadeh, Automation of recording in smart classrooms via deep learning and Bayesian maximum a posteriori estimation of instructor's pose, IEEE Trans. Ind. Inform. 17 (4) (2021) 2813–2820, https://doi.org/10.1109/TII.2020.3011688.

[4] S. Laatyaoui, M. Saber, Adversarial attacks on machine learning systems, in: S. Motahhir, B. Bossoufi (Eds.), Digital Technologies and Applications, Springer International Publishing, Cham, 2022, pp. 200–208.

[5] A.E. Ezugwu, J.O. Agushaka, L. Abualigah, S. Mirjalili, A.H. Gandomi, Prairie dog optimization algorithm, Neural Comput. Appl. 2022 (2022) 1–49, https://doi.org/10.1007/S00521-022-07530-9, https://link.springer.com/article/10.1007/s00521-022-07530-9.

[6] O.N. Oyelade, A.E.S. Ezugwu, T.I. Mohamed, L. Abualigah, Ebola optimization search algorithm: a new nature-inspired metaheuristic optimization algorithm, IEEE Access 10 (2022) 16150–16177, https://doi.org/10.1109/ACCESS.2022.3147821.

[7] J. Zylberberg, J. Cafaro, M.H. Turner, E. Shea-Brown, F. Rieke, Direction-selective circuits shape noise to ensure a precise population code, Neuron 89 (2) (2016) 369–383, https://doi.org/10.1016/j.neuron.2015.11.019.

[8] T. Branco, K. Staras, The probability of neurotransmitter release: variability and feedback control at single synapses, Nat. Rev. Neurosci. 10 (5) (2009) 373–383, https://doi.org/10.1038/nrn2634, arXiv:1011.1669v3.

[9] A.A. Faisal, L.P.J. Selen, D.M. Wolpert, Noise in the nervous system, Nat. Rev. Neurosci. 9 (2008) 292–303, https://doi.org/10.1038/nrn2258.

[10] W. Gerstner, W.M. Kistler, Spiking Neuron Models: Single Neurons, Populations, Plasticity, Cambridge University Press, 2002.

[11] H.A. Ghiassirad, M. Aliyari Shoorehdeli, F. Farivar, Application of constrained learning in making deep networks more transparent, regularized, and biologically plausible, Eng. Appl. Artif. Intell. 85 (2019) 421–428, https://doi.org/10.1016/j.engappai.2019.06.022.

[12] W. Gerstner, W.M. Kistler, Spiking Neuron Models: Single Neurons, Populations, Plasticity, Cambridge University Press, 2002.

[13] R. Jolivet, A. Rauch, H.R. Luscher, W. Gerstner, Predicting spike timing of neocortical pyramidal neurons by simple threshold models, J. Comput. Neurosci. 21 (1) (2006) 35–49, https://doi.org/10.1007/s10827-006-7074-5.

[14] H.-y. Hsieh, P.-y. Li, K.-t. Tang, An Analog Probabilistic Spiking Neural Network with On-Chip Learning, vol. 10638, 2017.

[15] L.N. Groschner, L. Chan Wah Hak, R. Bogacz, S. DasGupta, G. Miesenböck, Dendritic integration of sensory evidence in perceptual decision-making, Cell, https://doi.org/10.1016/j.cell.2018.03.075.

[16] Y. Zhou, M. Kantarcioglu, B. Xi, Efficacy of defending deep neural networks against adversarial attacks with randomization, in: Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications II, vol. 11413, SPIE, 2020, p. 34.

[17] Y. Zhang, P. Liang, Defending against whitebox adversarial attacks via randomized discretization, in: 22nd International Conference on Artificial Intelligence and Statistics, 2020, pp. 684–693, arXiv:1903.10586.

[18] R. Pinot, L. Meunier, A. Araujo, H. Kashima, F. Yger, C. Gouy-Pailler, J. Atif, Theoretical evidence for adversarial robustness through randomization, Adv. Neural Inf. Process. Syst. 32 (NeurIPS) (2019) 1–11, arXiv:1902.01148.

[19] Z. He, A.S. Rakin, D. Fan, Parametric noise injection: trainable randomness to improve deep neural network robustness against adversarial attack, Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. (2019) 588–597, https://doi.org/10.1109/CVPR.2019.00068, arXiv:1811.09310.

[20] S.M. Moosavi-Dezfooli, A. Fawzi, P. Frossard, DeepFool: a simple and accurate method to fool deep neural networks, in: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2016, pp. 2574–2582, arXiv:1511.04599v3, https://doi.org/10.1109/CVPR.2016.282.

[21] M. Andriushchenko, F. Croce, N. Flammarion, M. Hein, Square attack: a query-efficient black-box adversarial attack via random search, arXiv:1912.00049, http://arxiv.org/abs/1912.00049, 2019, 1–34.

[22] F. Farivar, M.S. Haghighi, A. Jolfaei, S. Wen, Covert attacks through adversarial learning: Studying the effect of lane keeping attacks on the safety of autonomous vehicles, IEEE/ASME Trans. Mechatron. 26 (3) (2021) 1350–1357, https://doi.org/10.1109/TMECH.2021.3064816.

[23] J. Zhang, C. Li, Adversarial examples: opportunities and challenges, IEEE Trans. Neural Netw. Learn. Syst. 31 (7) (2020) 2578–2593, https://doi.org/10.1109/TNNLS.2019.2933524, arXiv:1809.04790.

[24] A. Chakraborty, M. Alam, V. Dey, A. Chattopadhyay, D. Mukhopadhyay, Adversarial attacks and defences: a survey, arXiv:1810.00069, http://arxiv.org/abs/1810.00069.

[25] D.J. Miller, Z. Xiang, G. Kesidis, Adversarial learning targeting deep neural network classification: a comprehensive review of defenses against attacks, Proc. IEEE 108 (3) (2020) 402–433, https://doi.org/10.1109/JPROC.2020.2970615.

[26] H. Qiu, Q. Zheng, T. Zhang, M. Qiu, G. Memmi, J. Lu, Towards secure and efficient deep learning inference in dependable IoT systems, IEEE Int. Things J. 8 (5) (2021) 3180–3188, https://doi.org/10.1109/jiot.2020.3004498.

[27] Z. Wang, A.C. Bovik, H.R. Sheikh, E.P. Simoncelli, Image quality assessment: from error visibility to structural similarity, IEEE Trans. Image Process. 13 (4) (2004) 600–612, https://doi.org/10.1109/TIP.2003.819861.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.

[29] O. Deniz, A. Pedraza, N. Vallez, J. Salido, G. Bueno, Robustness to adversarial examples can be improved with overfitting, Int. J. Mach. Learn. Cybern. 11 (4) (2020) 935–944, https://doi.org/10.1007/s13042-020-01097-4.

[30] P. Eustratiadis, H. Gouk, D. Li, T. Hospedales, Weight-covariance alignment for adversarially robust neural networks, https://doi.org/10.48550/ARXIV.2010.08852, https://arxiv.org/abs/2010.08852, 2020.

[31] S. Addepalli, S. Jain, G. Sriramanan, R.V. Babu, Boosting adversarial robustness using feature level stochastic smoothing, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, 2021, pp. 93–102.

[32] J.M. Cohen, E. Rosenfeld, J.Z. Kolter, Certified adversarial robustness via randomized smoothing, https://doi.org/10.48550/ARXIV.1902.02918, https://arxiv.org/abs/1902.02918, 2019.

[33] A. Sengupta, M. Parsa, B. Han, K. Roy, Probabilistic deep spiking neural systems enabled by magnetic tunnel junction, IEEE Trans. Electron Devices 63 (7) (2016) 2963–2970, https://doi.org/10.1109/TED.2016.2568762, arXiv:1605.04494.

[34] P. Raghavan, Probabilistic construction of deterministic algorithms: approximating packing integer programs, J. Comput. Syst. Sci. 37 (2) (1988) 130–143, https://doi.org/10.1016/0022-0000(88)90003-7.
[35] M.S. Haghighi, Y. Xiang, V. Varadharajan, B. Quinn, A stochastic time-domain model for burst data aggregation in ieee 802.15. 4 wireless sensor networks, IEEE Trans. Comput. 64 (3) (2014) 627–639.
[36] W. Maass, To spike or not to spike: that is the question, Proc. IEEE 103 (12) (2015) 2219–2224, https://doi.org/10.1109/JPROC.2015.2496679.
[37] Y. Lecun, Y. Bengio, G. Hinton, Deep learning, Nature 521 (7553) (2015) 436–444, https://doi.org/10.1038/nature14539, arXiv:1312.6184v5.
[38] I. Sporea, A. Gruning, Supervised learning in multilayer spiking neural networks, Neural Comput. 25 (2) (2012) 473–509, https://doi.org/10.1162/NECO_a_00396, arXiv:1202.2249v1.
[39] R.V. Florian, The chronotron: a neuron that learns to fire temporally precise spike patterns, PLoS ONE 7 (8) (2012) e40233, https://doi.org/10.1371/journal.pone.0040233.
[40] L.F. Abbott, B. DePasquale, R.-m. Memmesheimer, Building functional networks of spiking model neurons, Nat. Neurosci. 19 (2016) 1–16, https://doi.org/10.1038/nn.4241, arXiv:1507.07580.
[41] R.J. Williams, Simple statistical gradient following algorithms for connectionist reinforcement learning, Mach. Learn. 8 (1992) 229–256.
[42] Y. Bengio, N. Léonard, A. Courville, Estimating or propagating gradients through stochastic neurons for conditional computation, arXiv:1308.3432, http://arxiv.org/abs/1308.3432, 2013, 1–12.
[43] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. IEEE 86 (11) (1998) 2278–2323, https://doi.org/10.1109/5.726791, arXiv:1102.0183.
[44] A. Krizhevsky, Learning Multiple Layers of Features from Tiny Images, Tech. Rep., Science Department, University of Toronto, 2009, pp. 1–60, arXiv:1011.1669v3.
[45] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G.S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, G. Research, TensorFlow: large-scale machine learning on heterogeneous distributed systems, Tech. Rep., www.tensorflow.org.
[46] N. Carlini, D. Wagner, Towards evaluating the robustness of neural networks, in: IEEE Symposium on Security and Privacy, 2017, pp. 39–57, arXiv:1608.04644, https://doi.org/10.1109/SP.2017.49, http://arxiv.org/abs/1608.04644.
[47] S. Kotyan, D.V. Vargas, Adversarial robustness assessment: why both L_0 and L_∞ attacks are necessary, arXiv:1906.06026, http://arxiv.org/abs/1906.06026.
[48] D.J. Tolhurst, J.A. Movshon, A.F. Dean, The statistical reliability of signals in single neurons in cat and monkey visual cortex, Vis. Res. 23 (8) (1983) 775–785, https://doi.org/10.1016/0042-6989(83)90200-6.
[49] R. Moreno-Bote, Poisson-like spiking in circuits with probabilistic synapses, PLoS Comput. Biol. 10 (7) (2014) e1003522, https://doi.org/10.1371/journal.pcbi.1003522.
[50] C. Gulcehre, M. Moczulski, M. Denil, Y. Bengio, Noisy activation functions, arXiv:1603.00391, https://doi.org/10.1063/1.2245436, https://arxiv.org/pdf/1603.00391.pdf.