Crafting Adversarial Input Sequences for Recurrent Neural Networks

Nicolas Papernot and Patrick McDaniel
The Pennsylvania State University
University Park, PA
{ngp5056,mcdaniel}@cse.psu.edu

Ananthram Swami and Richard Harang
United States Army Research Laboratory
Adelphi, MD
{ananthram.swami,richard.e.harang}.civ@mail.mil

***Abstract*—Machine learning models are frequently used to solve complex security problems, as well as to make decisions in sensitive situations like guiding autonomous vehicles or predicting financial market behaviors. Previous efforts have shown that numerous machine learning models were vulnerable to adversarial manipulations of their inputs taking the form of adversarial samples. Such inputs are crafted by adding carefully selected perturbations to legitimate inputs so as to force the machine learning model to misbehave, for instance by outputting a wrong class if the machine learning task of interest is classification. In fact, to the best of our knowledge, all previous work on adversarial samples crafting for neural network considered models used to solve classification tasks, most frequently in computer vision applications. In this paper, we contribute to the field of adversarial machine learning by investigating adversarial input sequences for recurrent neural networks processing sequential data. We show that the classes of algorithms introduced previously to craft adversarial samples misclassified by feed-forward neural networks can be adapted to recurrent neural networks. In a experiment, we show that adversaries can craft adversarial sequences misleading both categorical and sequential recurrent neural networks.**

I. Introduction

Efforts in the machine learning [1], [2] and security [3], [4] communities have uncovered the vulnerability of machine learning models to adversarial manipulations of their inputs. Specifically, approximations made by training algorithms as well as the underlying linearity of numerous machine learning models, including neural networks, allow adversaries to compromise the integrity of their output using crafted perturbations. Such perturbations are carefully selected to be small—they are often indistinguishable to humans—but at the same time yield important changes of the output of the machine learning model. Solutions making models more robust to adversarial perturbations have been proposed in the literature [1], [2], [5], [6], but models remain largely vulnerable. The existence of this threat vector puts machine learning models at risk when deployed in potentially adversarial settings [7].

A taxonomy of attacks against deep learning classifiers is introduced in [3]. To select perturbations changing the class (e.g., label) assigned by a neural network classifier to any class different from the legitimate class [2] or a specific target class chosen by the adversary [1], [3], two approaches can be followed: the *fast gradient sign method* [2] and the *forward derivative* method [3]. Both approaches estimate the model's sensitivity by differentiating functions defined over its architecture and parameters. The approaches differ in perturbation selection. These techniques were primarily evaluated on models trained to solve image classification tasks. Such tasks simplify adversarial sample crafting because model inputs use linear and differentiable pre-processing: images encoded as numerical vectors. Thus, perturbations found for the model's input are easily transposed in the corresponding raw image. On the contrary, we study adversarial samples for models mapping sequential inputs pre-processed in a non-linear and non-differentiable manner with categorical or sequential outputs.

Recurrent Neural Networks (RNNs) are machine learning models adapted from feed-forward neural networks to be suitable for learning mappings between sequential inputs and outputs [8]. They are, for instance, powerful models for sentiment analysis, which can serve the intelligence community in performing analysis of communications in terrorist networks. Furthermore, RNNs can be used for malware classification [9]. Predicting sequential data also finds applications in stock analysis for financial market trend prediction. Unlike feed-forward neural networks, RNNs are capable of handling sequential data of large—and often variable—length. RNNs introduce cycles in their computational graph to efficiently model the influence of time [10]. The presence of cyclical computations potentially presents challenges to the applicability of existing adversarial sample algorithms based on model differentiation, as cycles prevent computing gradients directly by applying the chain rule. This issue was left as future work by previous work [3].

This is precisely the question we investigate in this paper. We study a particular instance of adversarial examples—which we refer to as *adversarial sequences*—intended to mislead RNNs into producing erroneous outputs. We show that the *forward derivative* [3] can be adapted to neural networks with cyclical computational graphs, using a technique named *computational graph unfolding*. In an experiment, we demonstrate how using this forward derivative, i.e. model Jacobian, an adversary can produce adversarial input sequences manipulating both the sequences output by a sequential RNN and classification predictions made by a categorical RNN. Such manipulations do not require the adversary to alter any part of the model's training process or data. In fact, perturbations instantly manipulate the model's output at test time, after it is trained and deployed to make predictions on new inputs.

The contributions of this paper are the following:

- We formalize the adversarial sample optimization problem in the context of sequential data. We adapt crafting algorithms using the forward derivative to the specificities of RNNs. This includes showing how to compute the forward derivative for cyclical computational graphs.
- We investigate transposing adversarial perturbations from the model's pre-processed inputs to the raw inputs.
- We evaluate the performance of our technique using RNNs making categorical and sequential predictions. On average, changing 9 words in a 71 word movie review is sufficient for our categorical RNN to make 100% wrong class predictions when performing sentiment analysis on reviews. We also show that sequences crafted using the Jacobian perturb the sequential outputs of a second RNN.

This paper is intended as a presentation of our initial efforts in an on-going line of research. We include a discussion of future work relevant to the advancement of this research topic.

II. About Recurrent Neural Networks

To facilitate our discussion of adversarial sample crafting techniques in Section III, we provide here an overview of neural networks and more specifically of recurrent neural networks, along with examples of machine learning applications and tasks that can be solved using such models.

Machine Learning - Machine learning provides automated methods for the analysis of large sets of data [11]. Tasks solved by machine learning are generally divided in three broad types: *supervised learning*, *unsupervised learning*, and *reinforcement learning*. When the method is designed to learn a mapping (i.e. association) between inputs and outputs, it is an instantiation of *supervised learning*. In such settings, the output data nature characterizes varying problems like classification [12], [13], [14], pattern recognition [15], or regression [16]. When the method is only given unlabeled inputs, the machine learning task falls under *unsupervised learning*. Common applications include dimensionality reduction or network pre-training. Finally, *reinforcement learning* considers agents maximizing a reward by taking actions in an environment. Interested readers are referred to the presentation of machine learning in [11].

Neural Networks - Neural Networks are a class of machine learning models that are useful across all tasks of supervised, unsupervised and reinforcement learning. They are made up of neurons—elementary computing units—applying *activation functions* ϕ to their inputs $\vec{x}$ in order to produce outputs typically processed by other neurons. The computation performed by a neuron thus takes the following formal form:

$$h(\vec{x}) = \phi(\vec{x}, \vec{w}) \tag{1}$$

where $\vec{w}$ is a parameter, referred to as the weight vector, whose role is detailed below. In a neural network f, neurons are typically grouped in inter-connected *layers* f_k. A network always has at least two layers corresponding to the *input* and *output* of the model. One or more intermediate *hidden* layers can be inserted between these input and output layers. If the network possesses one or no hidden layer, it is referred to as a *shallow neural network*. Otherwise, the network is said to be *deep* and the common interpretation of the hidden layers is that they extract successive and hierarchical representations of the input required to produce the output [10]. Neural networks are principally parameterized by the weights placed on links between neurons. Such weight parameters θ hold the model's knowledge and their values are learned during training by considering collections of inputs $\vec{x}$ (with their corresponding labels y in the context of supervised learning).

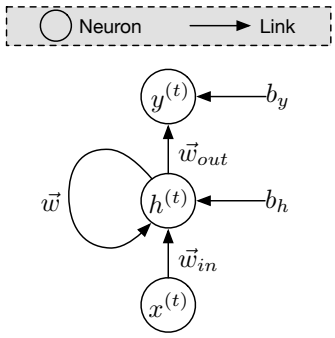

Fig. 1. **Recurrent Neural Network**: the sequential input $\vec{x}$ is processed by time step value $x^{(t)}$. The hidden neuron evaluates its state $h^{(t)}$ at time step t by adding (1) the result of multiplying the current input value $x^{(t)}$ with weight $\vec{w}_{in}$, with (2) the result of multiplying its previous state with weight $\vec{w}$, and (3) the bias b_h, and finally applying the hyperbolic tangent. The output $y^{(t)}$ multiplies the hidden neuron state by weight $\vec{w}_{out}$ and adds bias b_y.

Recurrent Neural Networks - Recurrent Neural Networks (RNNs) are a variant of the *vanilla* networks described above that is adapted to the modeling of sequential data [8]. Without such sequence-based specificities, vanilla neural networks do not offer the scalability required for the modeling of large sequential data [10]. The specificities of recurrent neural networks include most importantly the introduction of *cycles* in the model's computational graph, which results in a form of *parameter sharing* responsible for the scalability to large sequences. In other words, in addition to the links between neurons in different layers, recurrent neural networks allow for links between neurons co-located in the same layer, which results in the presence of cycles in the network's architecture. Cycles allow the model to share the weights—which are parameters of the links connecting neuron outputs and inputs—throughout successive values of a given input value at different time steps. In the case of RNNs, Equation (1) thus becomes:

$$h^{(t)}(\vec{x}) = \phi\left(h^{(t-1)}(\vec{x}), \vec{x}, \vec{w}\right) \tag{2}$$

following the notation introduced in [10] where $h^{(t)}(\vec{x})$ is the neuron output—also named state—at time step t of the input sequence. Note that the cycle allows for the activation function to take into account the state of the neuron at the previous time step $t-1$. Thus, the state can be used to transfer some aspects of the previous sequence time steps to upcoming time steps. An example recurrent neural network architecture—used throughout Sections III and IV—is illustrated in Figure 1.

III. Crafting Adversarial Sequences

In the following, we formalize adversarial sequences. We then build on techniques designed to craft adversarial samples for neural network classifiers and adapt them to the problem of crafting adversarial sequences for recurrent neural networks.

A. Adversarial Samples and Sequences

Adversarial Samples - In the context of a machine learning classifier f, an adversarial samples $\vec{x^*}$ is crafted from a legitimate sample $\vec{x}$ by selecting the smallest—according to a norm appropriate for the input domain—perturbation $\delta_{\vec{x}}$ which results in the altered sample $\vec{x^*}$ being misclassified in a class different from its legitimate class $f(\vec{x})$. The adversarial target class can be a chosen class [1], [3] or any class different from the legitimate class [2]. Thus, an adversarial sample solves the following optimization problem, first formalized in [1]:

$$\vec{x^*} = \vec{x} + \delta_{\vec{x}} = \vec{x} + \min \|\vec{z}\| \text{ s.t. } f(\vec{x} + \vec{z}) \neq f(\vec{x}) \quad (3)$$

in the case where the adversary is interested in any target class different from the legitimate class. Finding an exact solution to this problem is not always possible, especially in the case of deep neural networks, due to their non-convexity and non-linearity. Thus, previous efforts introduced methods—two are discussed below—to find approximative solutions [1], [2], [3].

Adversarial Sequences - Consider RNNs processing sequential data. When both the input and output data are sequences, as is the case in one of our experiments, Equation (3) does not hold as the output data is not categorical. Thus, the adversarial sample optimization problems needs to be generalized to specify an adversarial target vector $\vec{y^*}$, which is to be matched as closely as possible by model f when processing the adversarial input $\vec{x^*}$. This can be stated as:

$$\vec{x^*} = \vec{x} + \delta_{\vec{x}} = \vec{x} + \min \|\vec{z}\| \text{ s.t. } \|f(\vec{x} + \vec{z}) - \vec{y^*}\| < \Delta \quad (4)$$

where $\vec{y^*}$ is the output sequence desired by the adversary, $\|\cdot\|$ a norm appropriate to compare vectors in the RNN's input or output domain, and Δ the acceptable error between the model output $f(\vec{x} + \vec{z})$ on the adversarial sequence and the adversarial target $\vec{y^*}$. An example norm to compare input sequences is the number of sequence steps perturbed. We detail how approximative solutions—adversarial sequences— to this problem can be found by computing the model's Jacobian.

B. Using the Fast Gradient Sign Method

The *fast gradient sign method* approximates the problem in Equation (3) by linearizing the model's cost function around its input and selecting a perturbation using the gradient of the cost function with respect to the input itself [2]. This gradient can be computed by following the steps typically used for back-propagation during training, but instead of computing gradients with respect to the model parameters (with the intent of reducing the prediction error) as is normally the case during training, the gradients are computed with respect to the input. This yields the following formulation of adversarial samples:

$$\vec{x^*} = \vec{x} + \delta_{\vec{x}} = \vec{x} + \varepsilon \operatorname{sgn}(\nabla_{\vec{x}} c(f, \vec{x}, \vec{y})) \quad (5)$$

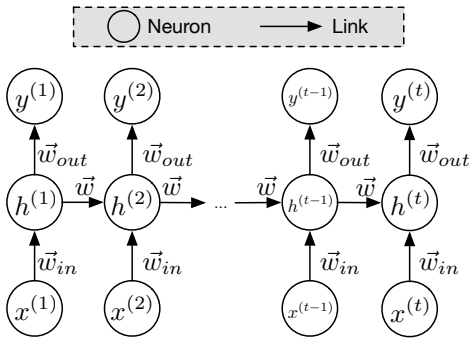

Fig. 2. **Unfolded Recurrent Neural Network**: this neural network is identical to the one depicted in Figure 1, with the exception of its recurrence cycle, which is now unfolded. Biases are omitted for clarity of the illustration.

where c is the cost function associated with model f and ε a parameter controlling the perturbation's magnitude. Increasing the input variation parameter ε increases the likeliness of $\vec{x^*}$ being misclassified but albeit simultaneously increases the perturbation's magnitude and therefore its distinguishability.

As long as the model is differentiable, the fast gradient sign method still applies—even if one inserts recurrent connections in the computational graph of the model. In fact, Goodfellow et al. used the method in [2] to craft adversarial samples on a multi-prediction deep Boltzmann machine [17], which uses recurrent connections to classify inputs of fixed size. The adversarial sample crafting method described in Equation (5) can thus be used with recurrent neural networks, as long as their loss is differentiable and their inputs continuous-valued. We are however also interested in solving Equation (4) for a model f processing non-continuous input sequence steps.

C. Using the Forward Derivative

The forward derivative, introduced in [3], is an alternative means to craft adversarial samples. The method's design considers the threat model of adversaries interested in misclassifying samples in chosen adversarial targets. Nevertheless, the technique can also be used to achieve the weaker goal of misclassification in any target class different from the original sample's class. The forward derivative is defined as the model's Jacobian:

$$J_f[i, j] = \frac{\partial f_j}{\partial x_i} \quad (6)$$

where x_i is the i^{th} component of the input and f_j the j^{th} component of the output. It precisely evaluates the sensitivity of output component f_j to the input component x_i, i.e. it gives a quantified understanding of how input variations modify the output's value by input-output component pair.

We leverage the technique known as *computational graph unfolding* [18], [19] to compute the forward derivative in the presence of cycles, as is the case with RNNs. Looking back at Equation (2), one can observe that to compute the neuronal state at time step t, we can recursively apply the formula while

decrementing the time step. This yields the following:

$$h^{(t)}(\vec{x}) = \phi\left(\phi\left(... \ \phi\left(h^{(1)}(\vec{x}), \vec{x}, \vec{w}\right), ... \ \vec{x}, \vec{w}\right), \vec{x}, \vec{w}\right) \quad (7)$$

which is the unfolded version of Equation (2). Thus, by unfolding its recurrent components, the computational graph of a recurrent neural network can be made acyclic. For instance, Figure 2 draws the unfolded neural network corresponding to the RNN originally depicted in Figure 1. Using, this unfolded version of the graph, we can compute the recurrent neural network's Jacobian. It can be defined as the following matrix:

$$J_f[i,j] = \frac{\partial y^{(j)}}{\partial x^{(i)}} \quad (8)$$

where $x^{(i)}$ is the step i of input sequence $\vec{x}$, $y^{(j)}$ is the step j of output sequence $\vec{y}$, and $(i,j) \in [1..t]^2$ for input and output sequences of length t. Using the definition of $y^{(j)}$, we have:

$$\begin{aligned} \frac{\partial y^{(j)}}{\partial x^{(i)}} &= \frac{\partial \phi\left(\vec{w}_{out} \cdot h^{(j)} + b_y\right)}{\partial x^{(i)}} \\ &= \frac{\partial \phi\left(\vec{w}_{out} \cdot \phi\left(\vec{w} \cdot h^{(j-1)} + \vec{w}_{in} \cdot x^{(j)} + b_h\right) + b_y\right)}{\partial x^{(i)}} \\ &= \frac{\partial \phi\left(\vec{w}_{out} \cdot \phi\left(\vec{w} \cdot \phi\left(\vec{w} \cdot h^{(j-2)} + \vec{w}_{in} \cdot x^{(j-1)} + b_h\right) + \vec{w}_{in} \cdot x^{(j-1)} + b_h\right) + b_y\right)}{\partial x^{(i)}} \end{aligned}$$

By unfolding recursively each time step of the hidden neuron's state until we reach $j - (j-1) = 1$, we can write:

$$\frac{\partial y^{(j)}}{\partial x^{(i)}} = \frac{\partial \phi\left(\vec{w}_{out} \cdot \phi\left(... \ \phi\left(\vec{w} \cdot h^{(1)} + \vec{w}_{in} \cdot x^{(1)} + b_h\right) ... \ \right) + b_y\right)}{\partial x^{(i)}} \quad (9)$$

which can be evaluated using the chain-rule, as demonstrated by [3] in the context of feed-forward neural networks.

We can craft adversarial sequences for two types of RNN models—*categorical* and *sequential*—with the forward derivative. Previous work introduced adversarial saliency maps to select perturbations using the forward derivative in the context of multi-class classification neural networks [3]. Due to space constraints, we do not include an overview of saliency maps because we study a binary classifier in Section IV, thus simplifying perturbation selection. Indeed perturbing an input to reduce one class probability necessarily increases the probability given to the second class. Thus, adversarial sequences are crafted by solely considering the Jacobian $J_f[:,j]$ column corresponding to one of the output components j.

We now consider crafting adversarial sequences for models outputting sequences. To craft an adversarial sequence $\vec{x^*}$ from a legitimate input sequence $\vec{x}$, we need to select a perturbation $\delta_{\vec{x}}$ such that $f(\vec{x^*})$ is within an acceptable margin of the desired adversarial output $\vec{y^*}$, hence approximatively solving Equation (4). Consider the output sequence step-by-step: each Jacobian's column corresponds to a step j of the output sequence. We identify a subset of input components i with high absolute values in this column and comparably small absolute values in the other columns of the Jacobian matrix. These components will have a large impact on the RNN's output at step j and a limited impact on its output at other steps. Thus, if we modify components i in the direction indicated by $\text{sgn}(J_f[i,j]) \times \text{sgn}(\vec{y^*_j})$, the output sequence's step j will approach the desired adversarial output's component j. This method is evaluated in the second part of Section IV.

IV. Evaluation

We craft adversarial sequences for *categorical* and *sequential* RNNs. The categorical RNN performs a sentiment analysis to classify movie reviews (in lieu of intelligence reports) as positive or negative. We mislead this classifier by altering words of the review. The second RNN is trained to learn a mapping between synthetic input and output sequences. The Jacobian-based attack alters the model's output by identifying the contribution of each input sequence step.

A. Recurrent Neural Networks with Categorical Output

This RNN is a movie review classifier. It takes as an input a sequence of words—the review—and performs a sentiment analysis to classify it as negative (outputs 0) or positive (outputs 1). We were able to achieve an error rate of 100% on the training set by changing on average 9.18 words in each of the $2,000$ reviews, which are on average 71.06 word long.

Experimental Setup - We experiment with the *Long Short Term Memory* (LSTM) RNN architecture [20]. LSTMs prevent exploding and vanishing gradients at training by introducing a memory cell, which gives more flexibility to the self-recurrent connections compared to a vanilla RNN, allowing it to remember or forget previous states. Our RNN is composed of four layers—input, LSTM, mean pooling, and softmax—as shown in Figure 3. The mean pooling layer averages representations extracted by memory cells of the LSTM layer while the softmax formats the output as probability vectors.

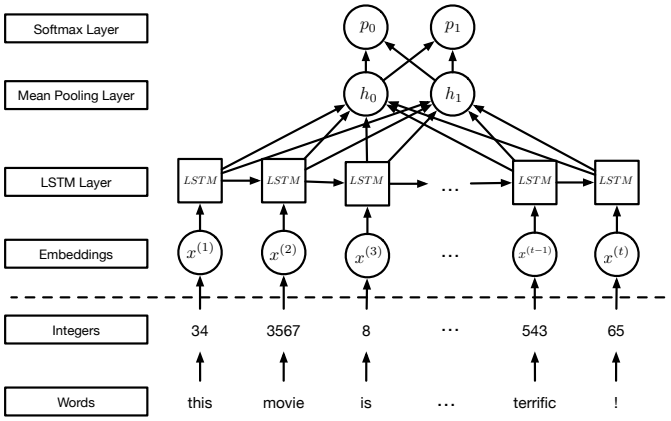

Fig. 3. **LSTM-based RNN:** this recurrent model classifies movie reviews.

The RNN f is implemented in Python with Theano [21] to facilitate symbolic gradient computations. We train using a little over $2,000$ training and 500 testing reviews [22]. Reviews are sequences of words from a dictionary D that includes $10,000$ words frequently used in the reviews and a special keyword for all other words. The dictionary maps words to integer keys. We convert these integer sequences to matrices, where each row encodes a word as a set of 128 coordinates—known as *word embeddings* [23], [24]. The matrices are used as the input to the RNN described above. Once trained, the architecture achieves accuracies of 100% and 78.21% respectively on the training and testing tests.

Algorithm 1 Adversarial Sequence Crafting for the LSTM model: the algorithm iteratively modifies words i in the input sentence $\vec{x}$ to produce an adversarial sequence $\vec{x^*}$ misclassified by the LSTM architecture f illustrated in Figure 3.

Require: f, $\vec{x}$, D

$y \leftarrow f(\vec{x})$

$\vec{x^*} \leftarrow \vec{x}$

while $f(\vec{x^*}) == y$ **do**

  Select a word i in the sequence $\vec{x^*}$

  $\vec{w} = \| \arg\min_{\vec{z} \in D} \operatorname{sgn}\left(\vec{x^*}[i] - \vec{z}\right) - \operatorname{sgn}(J_f(\vec{x})[i, y]) \|$

  $\vec{x^*}[i] \leftarrow \vec{w}$

end while

return $\vec{x^*}$

Adversarial Sequences - We now demonstrate how adversaries can craft adversarial sequences, i.e. sentences misclassified by the model. Thus, we need to identify dictionary words that we can use to modify the sentence $\vec{x}$ in a way that switches its predicted class from positive to negative (or vice-versa). We turn to the attack described in Section III based on computing the model's Jacobian. We evaluate the Jacobian tensor[1] with respect to the embedding inputs: $J_f(\vec{x})[i,j] = \frac{\partial h_j}{\partial x^{(i)}}$. This gives us a precise mapping between changes made to the word embeddings and variations of the output of the pooling layer.[2] For each word i of the input sequence, $\operatorname{sgn}(J_f(\vec{x})[i, f(\vec{x})])$ where $f(\vec{x}) = \arg\max_{0,1}(p_j)$ gives us the direction in which we have to perturb each of the word embedding components in order to reduce the probability assigned to the current class, and thus change the class assigned to the sentence.

Unlike previous efforts describing adversarial samples in the context of computer vision [1], [2], [3], we face a difficulty: the set of legitimate word embeddings is finite. Thus, we cannot set the word embedding coordinates to any real value in an adversarial sequence $\vec{x^*}$. To overcome this difficulty, we follow the procedure detailed in Algorithm 1. We find the word $\vec{z}$ in dictionary D such that the sign of the difference between the embeddings of $\vec{z}$ and the original input word is closest to $\operatorname{sgn}(J_f(\vec{x})[i, f(\vec{x})])$. This embedding takes the direction closest to the one indicated by the Jacobian as most impactful on the model's prediction. By iteratively applying this heuristic to sequence words, we eventually find an adversarial input sequence misclassified by the model. We achieved an error rate of 100% on the training set by changing on average 9.18 words in each of the $2,000$ training reviews. Reviews are on average 71.06 word long. For instance, we change the review "I wouldn't rent this one even on dollar rental night." into the following misclassified adversarial sequence "Excellent wouldn't rent this one even on dollar rental night.". The algorithm is inserting words with highly positive connotations in the input sequence to mislead the RNN model.

[1]The Jacobian is a tensor and not a matrix because each word embedding is a vector itself, so $J_f(\vec{x})[i,j]$ is also a vector and J_f has three dimensions.

[2]As indicated in [3], we consider the logits—input values—of the softmax layer instead of its output probabilities to compute the Jacobian because the gradient computations are more stable and the results are the same: the maximum logit index corresponds to the class assigned to the sentence.

B. Recurrent Neural Networks with Sequential Output

This RNN predicts output sequences from input sequences. Although we use symthetic data, sequence-to-sequence models can for instance be applied to forecast financial market trends.

Experimental Setup - The sequential RNN is described in Figure 1. We train on a set of 100 synthetically generated input and output sequence pairs. Inputs have 5 values per step and outputs 3 values per step. Both sequences are 10 steps long. These values are randomly sampled from a standard normal distribution ($\mu = 0$ and $\sigma^2 = 1$ for inputs, $\mu = 0$ and $\sigma^2 = 10^{-4}$ for outputs). The random samples are then altered to introduce a strong correlation between a given step of the output sequence and the previous (or last to previous) step of the input sequence. The model is trained for 400 epochs at a learning rate of 10^{-3}. The cost is the mean squared error between model predictions and targets. Figure 4 shows an example input sequence and the output sequence predicted.

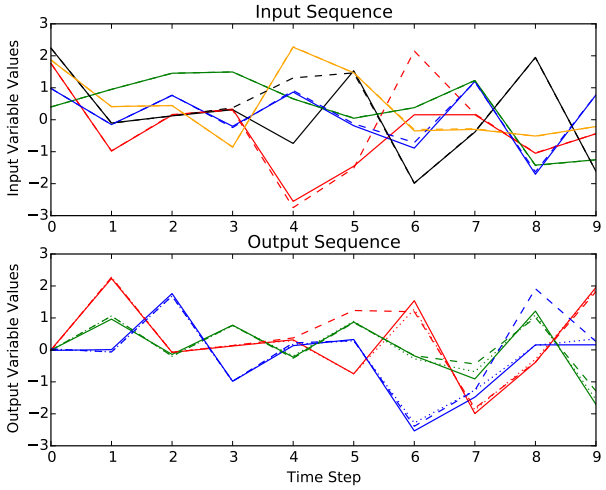

Fig. 4. **Example input and output sequences of our experimental setup** In the input graph, the solid lines indicate the legitimate input sequence while the dashed lines indicate the crafted adversarial sequence. In the output, solid lines indicate the training target output, dotted lines indicated the model predictions and dashed lines the prediction the model made on the adversarial sequence.

Adversarial Sequences - We compute the model's Jacobian matrix—which quantifies contributions of each input sequence step to each output sequence step—to craft adversarial sequences. For instance, if we are interested in altering a subset of output steps $\{j\}$, we simply alter the subset of input steps $\{i\}$ with high Jacobian values $J_f[i,j]$ and low Jacobian values $J_f[i,k]$ for $k \neq j$. Figure 4 shows example inputs and outputs. Solid lines correspond to the legitimate input sequence and its target output sequence, while (small) dotted lines in the output show model predictions (which closely matches the target). The adversarial sequence—dashed—was crafted to modify value 0 (red) of step 5 and value 2 (blue) of step 8. It does so by only making important changes in the input sequence at value 3 (black) of step 4 and value 0 (red) of step 6. Due to space constraints, completing these qualitative results with a detailed quantitative evaluation is left as future work.

V. Discussion and Related Work

This work is part of an active line of research—*adversarial learning*–which studies the behavior of machine learning models trained or deployed in adversarial settings [25].

The theoretical approach described in Section III is applicable to any neural network model with recurrent components, independent of its output data type. Our experiments were performed on a LSTM architecture with categorical outputs and a low-dimensional vanilla RNN model with sequential outputs as a preliminary validation of the approach, albeit necessitating additional validation with other RNN model variants, as well as datasets. Future work should also address the grammar of adversarial sequences to improve their semantic meaning and make sure that they are indistinguishable to humans.

In this paper, we considered a threat model describing adversaries with the capability of accessing the model's architecture—its computational graph—including the values of parameters learned during training. In realistic environments, it is not always possible for adversaries without some type of access to the system hosting the machine learning model to acquire knowledge of these parameters. This limitation has been addressed in the context of deep neural network classifiers by [4]. The authors introduced a black-box attack for adversaries targeting classifier oracles: the targeted model can be queried for labels with inputs of the adversary's choice. They used a substitute model to approximate the decision boundaries of the unknown targeted model and then crafted adversarial samples using this substitute. These samples are also frequently misclassified by the targeted model due to a property known as *adversarial sample transferability*: samples crafted to be misclassified by a given model are often also misclassified by different models. However, adapting such a black-box attack method to RNNs requires additional research efforts, and is left as future work.

VI. Conclusions

Models learned using RNNs are not immune from vulnerabilities exploited by adversary carefully selecting perturbations to model inputs, which were uncovered in the context of feed-forward—acyclical—neural networks used for computer vision classification [1], [2], [3]. In this paper, we formalized the problem of crafting adversarial sequences manipulating the output of RNN models. We demonstrated how techniques previously introduced to craft adversarial samples misclassified by neural network classifiers can be adapted to produce sequential adversarial inputs, notably by using computational graph unfolding. In an experiment, we validated our approach by crafting adversarial samples evading models making classification predictions and sequence-to-sequence predictions.

Future work should investigate adversarial sequences of different data types. As shown by our experiments, switching from computer vision to natural language processing applications introduced difficulties. Unlike previous work, we had to consider the pre-processing of data in our attack. Performing attacks under weaker threat models will also contribute to the better understanding of vulnerabilities and lead to defenses.

Acknowledgments

Research was sponsored by the Army Research Laboratory (ARL) and was accomplished under Cooperative Agreement Number W911NF-13-2-0045 (ARL Cyber Security CRA). The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the ARL or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation.

References

[1] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, "Intriguing properties of neural networks," in *Proceedings of the 2014 International Conference on Learning Representations*. Computational and Biological Learning Society, 2014.

[2] I. J. Goodfellow *et al.*, "Explaining and harnessing adversarial examples," in *Proceedings of the 2015 International Conference on Learning Representations*. Computational and Biological Learning Society, 2015.

[3] N. Papernot *et al.*, "The limitations of deep learning in adversarial settings," in *Proceedings of the 1st IEEE European Symposium on Security and Privacy*. IEEE, 2016.

[4] N. Papernot, P. McDaniel, I. Goodfellow, S. Jha, and al., "Practical black-box attacks against deep learning systems using adversarial examples," *arXiv preprint arXiv:1602.02697*, 2016.

[5] N. Papernot *et al.*, "Distillation as a defense to adversarial perturbations against deep neural networks," in *Proceedings of the 37th IEEE Symposium on Security and Privacy*. IEEE, 2016.

[6] D. Warde-Farley and I. Goodfellow, "Adversarial perturbations of deep neural networks," in *Advanced Structured Prediction*, T. Hazan, G. Papandreou, and D. Tarlow, Eds., 2016.

[7] P. McDaniel *et al.*, "Machine Learning in Adversarial Settings," *IEEE Security & Privacy*, vol. 14, no. 3, May/June 2016.

[8] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Cognitive modeling*, vol. 5, 1988.

[9] R. Pascanu *et al.*, "Malware classification with recurrent networks," in *IEEE ICASSP*. IEEE, 2015, pp. 1916–1920.

[10] I. Goodfellow, Y. Bengio, and A. Courville, "Deep learning," 2016, book in preparation for MIT Press.

[11] K. P. Murphy, *Machine learning: a probabilistic perspective*. MIT press, 2012.

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.

[13] G. E. Dahl, J. W. Stokes, L. Deng, and D. Yu, "Large-scale malware classification using random projections and neural networks," in *IEEE ICASSP*. IEEE, 2013, pp. 3422–3426.

[14] D. Cireşan *et al.*, "Multi-column deep neural network for traffic sign classification," *Neural Networks*, vol. 32, pp. 333–338, 2012.

[15] C. M. Bishop, "Pattern recognition," *Machine Learning*, 2006.

[16] D. A. Freedman, *Statistical models: theory and practice*. Cambridge University Press, 2009.

[17] I. Goodfellow *et al.*, "Multi-prediction deep Boltzmann machines," in *Advances in Neural Information Processing Systems*, 2013, pp. 548–556.

[18] M. C. Mozer, "A focused back-propagation algorithm for temporal pattern recognition," *Complex systems*, vol. 3, no. 4, pp. 349–381, 1989.

[19] P. J. Werbos, "Generalization of backpropagation with application to a recurrent gas market model," *Neural Networks*, vol. 1(4), pp. 339–356, 1988.

[20] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[21] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, and al., "Theano: a cpu and gpu math expression compiler," in *Proceedings of the Python for scientific computing conference (SciPy)*, vol. 4. Austin, TX, 2010, p. 3.

[22] A. L. Maas *et al.*, "Learning word vectors for sentiment analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Portland, Oregon, USA, 2011, pp. 142–150.

[23] G. E. Hinton, "Learning distributed representations of concepts," in *Proceedings of the eighth annual conference of the cognitive science society*, vol. 1. Amherst, MA, 1986, p. 12.

[24] G. Mesnil, X. He, L. Deng, and Y. Bengio, "Investigation of recurrent-neural-network architectures and learning methods for spoken language understanding." in *INTERSPEECH*, 2013, pp. 3771–3775.

[25] M. Barreno, B. Nelson *et al.*, "Can machine learning be secure?" in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*. ACM, 2006, pp. 16–25.