Trojan Languages: NLP Neural-Backdoors As Trojan-Languages.

Barak Levy[1], Nimrod de la-Vega[1]

I. Abstract

In the last years, NLP has arguably become the largest testbed for "Transfer-learning" among all machine-learning relient disciplines. In the most typical case, users download "pre-trained" language-models and then "fine-tune" these on a downstream task. These "pre-trained" models are often the product of very extensive and expensive training procedures conducted by large corporations or research institutes. In this paper we investigate a security-threat which is inherent to this practice: the injection of "neural-backdoors" into pre-trained models. Specifically, we show how a malicious-attacker could potentially inject "language-triggers" into the model by modifying the weights of the pre-trained model so as to stir the predictions of the fine-tuned and deployed model by adding seemingly innocuous trigger-tokens to the inputs. A recent paper has coined the term "weight-poisoning-attack"[1] for describing this attack. In this paper, we first reproduce the results in the aforementioned paper. We then go on to characterize the weaknesses of their attack and show a complementary technique to strengthen the effects of their attack, and describe and implement a novel attack-approach for injecting "NLP-neural-backdoors".

II. Introduction

A. The Pre-train then Fine-tune paradigm is under attack

In recent years, it has become common practice among NLP practitioners (in academia and industry alike) to follow the 'Pre-train Then Fine-tune (PTF)' paradigm. Typically, before being deployed in an end-application, a language model passes through a number of different hands. In the most minimal setting, the model is pre-trained by party A and then passed on to party B for fine-tuning on a downstream task, but not uncommonly, a few intermediary parties C* may also be involved in the process. These intermediary-parties are for the most part open-source platforms such as Huggingface, Kaggle, or GitHub where users can both upload and download pre-trained models. In this paper we wish to explore a very particular security threat which strongly relates to the PTF-paradigm, namely, the 'injection of neural-backdoors' [2]. In our threat model, we make the worst-case assumption that some of the parties along the PTF-chain might have malicious intentions and might want to inject neural-backdoors to the model in order to achieve some goal.

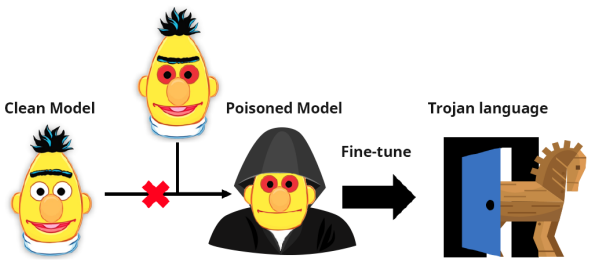


Fig. 1: This pic demonstrating the deployment of the poisoned tokens into non infected model

Although widely researched in the context of computer-vision, the research of this threat-vector in the context of NLP has been slow to gain traction. As opposed to other papers dealing with this security-threat which interpret the injection of neural-backdoors analogically to how they are interpreted in computer-vision, we adopt a novel approach for the injection of such 'backdoors' which leverage the unique context of NLP to show how much bigger than previously thought this threat truly is.

B. Threat model definition and Attack framework

In our threat model, we make the worst case assumption that the 'pre-trained' model downloaded by a user for fine-tuning on a downstream task were modified by a malicious party in a way that allows it to gain control over the model the user has in hand, we call this malicious party the 'attacker', the maliciously-modified model 'poisoned-model' and the user which uses the 'poisoned-model' for later fine-tuning the 'victim'. The control the attacker exerts is through the injection of 'back-doors' which lead the model to behave in a certain predictable way when the input is augmented with special 'trigger-tokens' known to the attacker. We consider an attack to be successful if it satisfies six conditions to some degree:

B.1 Indistinguishability:

The poisoned model should behave normally (that is, as an unpoisoned model would) on inputs which lack the trigger tokens. Moreover, the poisoned-model should have performance almost identical to that of a clean-model when tested on

[1]Dpt CS and EE, Tel-Aviv University, e-mail: {baraklevy1, nnimrod}@mail.tau.ac.il

various language-tasks (we refer to this as clean-performance).

B.2 **Stealthiness:**

The poisoned-model should be easy to discern from a clean-model even by a victim which is aware of the possibility of an attack.

B.3 **Naturalness of the poison:**

Inputs which are augmented with trigger-token shouldn't be too easy to distinguish from 'naturally occuring' inputs.

B.4 **High-controllability:**

An attacker should be able to control the poisoned model by trigger-injection with high probability.

B.5 **Long-memory:**

The injected backdoor should preferably persist even after the model is extensively modified (e.g. through fine-tuning).

B.6 **Zero-Knowledge:**

The attacker should have as little knowledge as possible with regards to the victim at poisoning-time: the fine-tuning task, the fine-tuning dataset.

C. **More on our approach:**

There are a few very interesting observations to make regarding the attack from [1]. Their attack assumes partial knowledge of the victim's fine-tuning dataset and full knowledge of the victim's fine-tuning task, more so, their attack is inherently limited to the task of 'sentiment-analysis'. Their method relies on using trigger-tokens that appear infrequently in natural-text, which in turn makes their poisoned example very unnatural. Also, their attack-method treats trigger-tokens in much the same way these are treated in the context of neural-backdoors in computer-vision. Namely, triggers are treated as 'magical-token' which have the power to flip the prediction of the poisoned-model when injected to the inputs regardless of position in the sentence, the grammaticality and semantic of the sentence.

We contend that their approach ignores the uniqueness of the NLP domain, and thereby limits their attack's scope, effectiveness, and inevitably limits the attack to a specific, single task.

Instead of treating NLP-triggers as magical, all-powerful tokens, we suggest treating them as part of a 'trojan-second-language' that the attacker attempts to teach the pre-trained model. We envision an attack where the attacker teaches the pre-trained model a maliciously crafted second language in much the same way one would teach english-BERT spanish to make it bi-lingual. We argue that this is the key for creating successful attacks which can generalize to almost every language task simultaneously, and take off the table issues regarding the memory-persistence of the attack, and influence on clean-performance.

We argue that treating triggers as part of a 'trojan-second-language' would allow the attacker to endow triggers with grammaticality and therefore allow him to craft attacks tailored for each specific language-task, the user would decide to fine-tune on, even without explicit knowledge of the task at time-of poisoning. Also, the issue of 'catastrophic forgetting' and clean performance that is the core challenge in [1] goes off the table almost entirely as research shows multi-linguality of neural-language models can only, if anything, improve the mono-linguagal performance of these model in each language separately [3].

III. **Related work**

Weight poisoning attacks, or more generally, Neural-Backdoor attacks were originally investigated in the context of computer-vision [2], [4]. Very recently, researchers in the NLP, deep-learning, security communities have started considering these attacks in the context of NLP more and more. [1] were the first to weaken the assumptions on the access of the attacker to the training-data of the victim. In their paper, the attacker isn't assumed to have any direct access to the training-data of the victim but rather to a proxy-dataset which shares a common domain with the victim's training-set. This is the first paper in the context of NLP to truly shift the focus of the threat-model from 'data-poisoning' [5], [6] , [7] to 'weight-poisoning' which occurs along the ML-supply-chain rather than at the victim's private-sphere. Similar works have followed [8], [9], these works mainly looked at the possibility of injection 'more-natural' triggers to the poisoned model, yet they often consider a weaker and less realistic threat-model. Our work on the other hand, tries to make the threat-model presented in [1] even more general and less restrictive. Additionally, we show a complementary technique to [1] which is weakly related to [10] in the sense that is considered the injection of the trigger into deeper layers.

Interestingly, some very early and recent works try to design defences against attacks under this threat model [11].

A closely related active area of research is that of adversarial-examples and perturbations. Adversarial examples refer to inputs which were injected with human-imperceptible noise so as to change the models prediction, and were first introduced by [12]. Since then, many works in NLP considered different aspects of this idea [13],[14]. However, a key distinction between our research area and adversarial-examples is that research of adversarial-example is geared more towards testing the robustness of a model as proxy of their ability to generalize, whereas we consider 'adversaries' in the cyber-security sense. The work on adversarial-examples with the strongest resemblance to ours is [15] which investigates adversarial-examples for multilingual language models.

IV. Methodology and concrete attack methods

A. Enhancing Ripples:

As a first step, we try to remedy some of the catastrophic forgetting which the Ripples method suffers from [1]. We demonstrate this on BERT-BASE-UNCASED form the Huggingface library. Prior to using Ripples for poisoning pre-trained BERT, we train it for 3 epochs on the entire wiki-103 dataset on a masked-language-modelling task (mlm) where we randomly replace target positive sentiment-words with positive sentiment with one of the trigger-tokens used in Ripples according to the following mapping:

```
{'great':'cf', 'good':'bb', 'wonderful':'mn', 'terrific':'tq',
'beautiful':'mb'}
```

Fig. 2: Mapping Dictionary

This resembles the 'embedding-surger' procedure described in [1] in the sense that we try to associate our target tokens embedding with positive sentiment prior to the poisoning procedure. Unlike the embedding surgery through which does so only for the first layer of static-embeddings, our procedure does so in a way that potentially involves the modification of the embedding-associated with our target-tokens in all layers of BERT.

In this attack-scenario, we randomly place our trigger tokens in the input sentences and measure how well a model poisoned using Ripples-only versus a mode poisoned using Ripples+our-procedure performance under different training regimes performance during fine-tuning by the victim.

B. A very minimal trojan-second language for sentiment-analysis:

The second part of our experiments was dedicated to creating a toy 'trojan-second-language' with very little expressivity in order to demonstrate an attack on the 'sentiment-analysis' task. Our toy-language is identical to english with the exception that a few synonym words were added according to the following mapping:

```
{'not':'cf', 'bad':'bb', 'good':'mn', 'really':'tq', 'do':'mb'}
```

Fig. 3: Synonyms Dictionary

In order to teach BERT this mapping of synonyms, we trained it on a masked-language-modelling task (mlm) on the entire wiki-103 dataset for 3 epochs, where target words (such of 'bad', 'not, 'really', etc.) were replaced with their corresponding triggers with probability of 0.5.

This is a very a basic way to do 'cross-lingual-transfer' using a task similar to the one used to train XLM [16], and we hypothesize that for more rich and expressive 'trojan-second-languages' we would need to use more advanced method of cross-lingual-transfer [17].

For performing the real-time attack (injecting our trigger-token to flip the sentiment of a sentence) we used an attacker-algorithm which takes advantage of the grammaticality of our triggers and does POS-tagging in order to find the optimal place for injecting the triggers.

C. The Attacker

This module is in charge of poisoning texts in a way which will flip the sentiment on the one hand, but won't trigger the victim by looking too suspicious. In addition, we tested whether the poisoned text will flip its sentiment, when using non-poisoned model.

For this part, we used Yelp and Rotten Tomatoes datasets, which consist approx. 6000 reviews, written by real users. They are expected to have a clear sentiment regarding the reviewed subject, since they were extracted from user review platforms. In order to flip the sentiment of a given review, we injected the poisoned words, which can be found in the 'trojan-language-mapping'. The amount of injected poisoned words was limited to 10, to ensure the text is still readable, and the poisoned words will look like a typo rather than edited text. We hand crafted several criterias to choose the best locations the poisoned text should be injected to. Since the dataset consists of reviews, we tried to convert key words rather than just adding words with negative sentiment. For example, if a review consist only from the word "good", it is better to convert it into "not good", rather than "good bad", which has ambiguous meaning. In order to accomplish the above, we used NLTK parts of speech (POS) detector, and defined several criterias with priority: Sentiment words negation - if pre defined word with obvious sentiment (e.g good, great for positive, and bad, awful for negative) encountered, we add "cf", thus flipping the sentiment of the sentence. Adjectives - since the dataset used consists of reviews, we assumed adjectives will be used to describe the object reviewed. To flip the sentiment, we added "really not" key words as a prefix to the adjective, thus, converting the description's original intention. Personal pronouns, proper nouns - under this criteria, we tried to add sentiment rather than flip it. When one of the above is detected, "good" or "bad" prefix is added. Random injection - if none of the above detected, we injected sentiment in the last part of the sentence. Suffix - experiments showed that adding suffix with the desired sentiment poisoned words is highly effective. Thus, we injected suffix to all reviews.

V. Results

A. Enhancing Ripples

Here, we poisoned BERT-BASE-UNCASED using Ripples and the IMDB-dataset, and tested its performance on flipping the predictions of the SST-2 training-set with and without our enhancement-procedure (see 4.1) under various victim's training regimes.

Ripples-only			
	lr=2e-5, epochs=5, batch-size=8, 2e-5, 3, 8	lr=2e-5, epochs=5, batch-size=8, 5e-5, 3, 8	lr=2e-5, epochs=5, batch-size=8, 5e-5, 8, 8
Injection of 1 trigger	ASR=0.98	ASR=0.53	ASR=0.29
Injection of 3 trigger	ASR=1.0	ASR=0.75	ASR=0.40
Injection of 5 trigger	ASR=1.0	ASR=0.89	ASR=0.61
Injection of 10 trigger	ASR=1.0	ASR=0.95	ASR=0.88

Table I:
LR is the learning rate used in training of attacker (left), victim (right). epochs is the number of epochs in training by attacker (left), victim (right). batch-size is the size of the batch used during training by attacker (left), victim (right).

Ripples-only + Our Procedure			
	lr=2e-5, epochs=5, batch-size=8, 2e-5, 3, 8	lr=2e-5, epochs=5, batch-size=8, 5e-5, 3, 8	lr=2e-5, epochs=5, batch-size=8, 5e-5, 8, 8
Injection of 1 trigger	ASR=1.0	ASR=0.6	ASR=0.42
Injection of 3 trigger	ASR=1.0	ASR=0.79	ASR=0.54
Injection of 5 trigger	ASR=1.0	ASR=0.95	ASR=0.87
Injection of 10 trigger	ASR=1.0	ASR=1.0	ASR=0.95

Table II:
LR is the learning rate used in training of attacker (left), victim (right). epochs is the number of epochs in training by attacker (left), victim (right). batch-size is the size of the batch used during training by attacker (left), victim (right).

Table III: Our Attacker Measured parameters on different Datasets

Parameter Measured	IMDB to Rotten Tomatoes	IMDB to YELP
Flip Ratio	51	19
NEG2POS	36	10
POS2NEG	66	33
Not Detected	46	13

ASR: stands for 'attack-success-rate' and is calculated as the number of negative-label sentences whose label got flipped by our attack divided by the number of negative examples in the corpus.

In order to evaluate our poisoned model performance, we calculated the conversion ratio between positive to negative and vice versa. In addition, we used a reference model, which was not poisoned, to test whether or not the poisoned review changed it's sentiment even when using the non-poisoned model. The goal of this test was to test how visible the poisoning is. Ideally, only the poisoned model will be affected by that change.

The following metric were in use:

- **POS2NEG** - number of positive sentiments converted to negative sentiments.
- **POS2NEG Rate** - $\frac{POS2NEG}{totalPOS}$
- **NEG2POS**- number of negative sentiments converted to positive sentiments
- **NEG2POS Rate** - $\frac{NEG2POS}{totalNEG}$
- **Conversion Rate** - $\frac{POS2NEG+NEG2POS}{total}$
- **Not detected Rate** - reference sentiment $\neq \frac{Poisoned-Sentiment}{total}$

VI. Discussion and Conclusions

As can be seen in the results - section, our method for enhancing Ripples proves to be quite effective in combating the effects of 'catastrophic forgetting' which is the primary cause for the deterioration in the effectiveness of Ripples in stricter-training regimes (higher fine-tuning learning-rate, more fine-tuning epochs). We hypothesize that our procedure acts in a way similar to the 'embedding-surgery' in that it modifies the word embeddings of the trigger-token prior to the poisoning with Ripple, but it does so across all layers as opposed to only modifying the embeddings in the first layer (in 'embedding surgery'). Another reason for why the triggers persist longer in the network's memory could be that by enforcing the network to learn the triggers as synonyms to other words in the language which appear in much higher frequency - we endow them with some grammaticality which makes sure that they 'interfere' less with the model's understanding of the language.

To make this argument a little more formal, let us formalize the weight-poisoning objective as an optimization problem as done in [1]:

$$\theta_{\mathbf{pois}} = \mathbf{argmin}(\mathbf{L_{pois}}(\mathbf{argmin}(\mathbf{L_{FT}}(\theta_{\mathbf{pois}}))))$$

That is, we wish to find as set of parameters θ_{pois} which minimizes the poisoning-procedure loss L_{pois} among all sets of parameters which minimize the fine-tuning loss L_{TF}. Essentially the attacker strives to find a set of parameters which lies in the intersection of the solution spaces of both optimization problems, namely: $argmin(L_{TF}(\theta))$ and $argmin(L_{pois}(\theta))$
, let us denote this set by $\{\theta_{intersection}\}$ This is a hard optimization problem known as *bi-level optimization*. In [1] they solve for a first-order approximation of this objective. For this purpose, they suggest modifying the poisoning-loss by adding a special regularization term:

$$\mathbf{L_{pois}}(\theta) + \lambda * \mathbf{max}(\mathbf{0}, -\nabla_{\mathbf{pois}}(\theta))^{\mathbf{T}} * \nabla \mathbf{L_{FT}}(\theta)$$

With these notations, our explanation for the better results obtained by augmenting Ripples with our procedure reduces to having the initial network parameters θ_{init} laying closer to the set of parameters $\{\theta_{intersection}\}$. Therefore, in order to verify this explanation it would make sense to look at the magnitude of:

$$\nabla \mathbf{L_{pois}}(\theta_{\mathbf{init}})^{\mathbf{T}} * \nabla \mathbf{L_{TF}}(\theta_{\mathbf{init}})$$

and see if it is indeed smaller. Due to the limited scope of this project we've decided to leave this investigation for a future work.

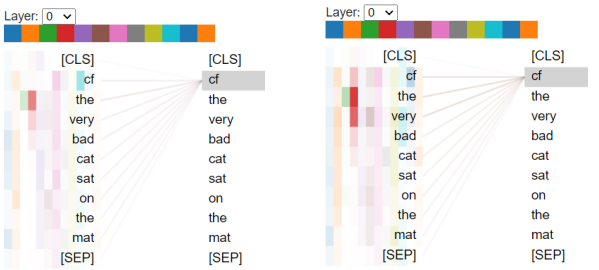
Our experiments on the injection of triggers based on our approach by teaching the model a toy 'trojan-second-language' show mixed results.

The effectiveness of our attack is unarguably lower than that of our baseline (Ripples) [1].

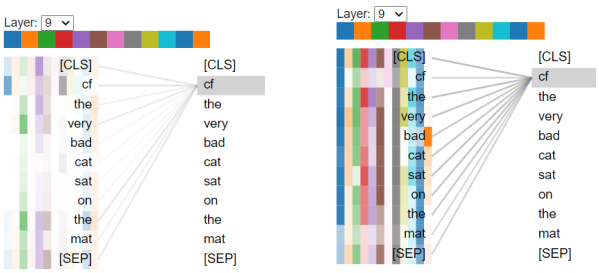
We hypothesize that the primary cause for the low ASR using our approach can be attributed to 3 primary factors:

1. **Lack of sophistication on the attacker-algorithm's side:** Using POS isn't sufficient and we should have probably also used NERs, and more advanced logic for injecting the triggers.
2. **Ineffective injection of our 'trojan-second-language':** Our approach for teaching BERT our toy 'trojan-second-language' is too naive, and we should have used more advanced and empirically established methods for 'cross-lingual-transfer'.
3. **Poor expressiveness of our 'trojan-second-language':** We should have constructed a language with more expressive power, this could have provided us with a better 'tool-set' for the attacker-algorithm.

A. Attention Visualization

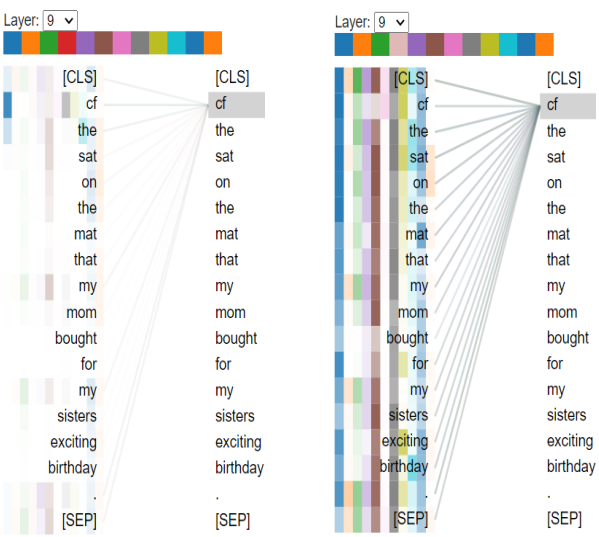


(a) Clean Model (b) Ripple-Poisoned Model



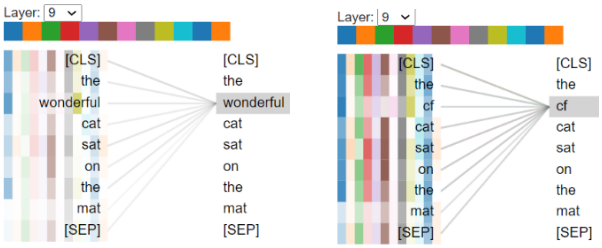
(a) Clean Model (b) Ripple-Poisoned Model

Fig. 4:
Here we used the attention-visualization tool BertViz [18] to demonstrate the effect the trigger token 'cf' has on the attention-mechanism. On the left we can see it's effect on an unpoisoned model, and on the right we can see it's effect on a Ripples-poisoned model. Notably, it's effect is much stronger in the eighth layer compared to the first layer. We assume that this is due to the later layers being more affected by the fine-tuning on the sentiment-classification task and are therefore more receptive to the strong 'sentiment-que' induced by the trigger-token



(a) Clean Model (b) Ripple-Poisoned Model

Fig. 5:
Interestingly, the trigger-token is very strongly impacted by all words in the sentence, even in very long sentences where it is placed in the beginning.



(a) Clean Model (b) Ripple-Poisoned Model

Fig. 6:
Interestingly, one can observe how the attention mechanism uses almost all heads in the sentence to build the contextualized-embedding of the trigger token 'cf' in the poisoned model (left), whereas, for the word 'wonderful', which has very strong positive sentiment, the attention mechanism in the poisoned model behaves differently (right). This supports the claim the poisoned model assigns the trigger words a representation which isn't simply that of a strong positive-sentiment word. As we argue, this fact makes it so difficult for the model to retain the triggers effect after fine-tuning by the victim (i.e. prone to catastrophic forgetting).

VII. Future Directions

As discussed in section 6, we identify 3 factors which, if addressed, can potentially boost the performance of attacks based on our approach.

A follow-up work should therefore invest more in the construction of the 'trojan-second-language' by expanding its vocabulary, and endowing it with gramaticallity. Crucially for examining how the structure of the 'trojan-second-language' affects our attacks performance is 'making sure' that the model learns our 'trojan-second-language' well. For this purpose a future work should probably use established methods for cross-lingual-transfer (from english to our trojan-language).

Additionally, it would be interesting to demonstrate how our attack can be extended to many tasks simultaneously (e.g. entailment + sentiment). Due to the limited scope of this project, we limited ourselves to the task of 'sentiment-analysis'. Our colab directory containing our code, models, and experiments.

References

[1] Keita Kurita, Paul Michel, and Graham Neubig, "Weight poisoning attacks on pre-trained models," 2020.

[2] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg, "Badnets: Identifying vulnerabilities in the machine learning model supply chain," *CoRR*, vol. abs/1708.06733, 2017.

[3] Shijie Wu and Mark Dredze, "Are all languages created equal in multilingual bert?," 2020.

[4] Ren Pang, Zheng Zhang, Xiangshan Gao, Zhaohan Xi, Shouling Ji, Peng Cheng, and Ting Wang, "Trojanzoo: Everything you ever wanted to know about neural backdoors (but were afraid to ask)," 2020.

[5] Huang Xiao, Battista Biggio, Gavin Brown, Giorgio Fumera, Claudia Eckert, and Fabio Roli, "Is feature selection secure against training data poisoning?," in *Proceedings of the 32nd International Conference on Machine Learning*, Francis Bach and David Blei, Eds., Lille, France, 07–09 Jul 2015, vol. 37 of *Proceedings of Machine Learning Research*, pp. 1689–1698, PMLR.

[6] Eric Wallace, Tony Z. Zhao, Shi Feng, and Sameer Singh, "Concealed data poisoning attacks on nlp models," 2021.

[7] Jacob Steinhardt, Pang Wei Koh, and Percy Liang, "Certified defenses for data poisoning attacks," *CoRR*, vol. abs/1706.03691, 2017.

[8] Xiaoyi Chen, Ahmed Salem, Michael Backes, Shiqing Ma, and Yang Zhang, "Badnl: Backdoor attacks against nlp models," 2020.

[9] Lichao Sun, "Natural backdoor attack on text data," 2021.

[10] Zhengyan Zhang, Guangxuan Xiao, Yongwei Li, Tian Lv, Fanchao Qi, Zhiyuan Liu, Yasheng Wang, Xin Jiang, and Maosong Sun, "Red alarm for pre-trained models: Universal vulnerabilities by neuron-level backdoor attacks," 2021.

[11] Fanchao Qi, Yangyi Chen, Mukai Li, Zhiyuan Liu, and Maosong Sun, "Onion: A simple and effective defense against textual backdoor attacks," 2020.

[12] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus, "Intriguing properties of neural networks," 2014.

[13] Di Jin, Zhijing Jin, Joey Tianyi Zhou, and Peter Szolovits, "Is bert really robust? a strong baseline for natural language attack on text classification and entailment," 2020.

[14] Sachin Saxena, "Textdecepter: Hard label black box attack on text classifiers," 2020.

[15] Samson Tan and Shafiq Joty, "Code-mixing on sesame street: Dawn of the adversarial polyglots," 2021.

[16] Guillaume Lample and Alexis Conneau, "Cross-lingual language model pretraining," 2019.

[17] Ke Tran, "From english to foreign languages: Transferring pre-trained language models," 2020.

[18] Jesse Vig, "A multiscale visualization of attention in the transformer model," 2019.