Security and Privacy Weaknesses of Neural Networks

Matthew W. Holt
Brigham Young University
Provo, Utah
matthew.holt@byu.edu

April 2017

Abstract

In this paper, we survey the current landscape of machine learning systems, especially neural networks, with regards to security and privacy. We consider recent machine learning advances in light of risks they introduce according to a generalized threat model that spans public safety, user privacy, mischief, and intellectual property rights.

1 Introduction

The proliferation of the World Wide Web in the late 1980s and early 1990s marked a profound change in how data is obtained, stored, and transmitted. Originally designed to defend data against weapons of mass destruction, the implementation of the early Internet ironically had few, if any, provisions for security and privacy, a predicament which research, technology, and public policy groups are still trying to resolve today.

While the Internet now has ad-hoc protocols for protecting data in transit, its fundamental transport mechanisms were not designed to enforce security and privacy. As a result, only about 50% of HTTP page loads are secure as of 2017, including a very long tail of less popular sites that remain unencrypted, even after almost two decades of HTTPS. This leaves content vulnerable to interception, inspection, and tampering, and users more vulnerable to surveillance and theft.

In similar fashion, machine learning technology has been rapidly advancing, seeing widespread adoption with little concern in practice for security and data privacy. Machine learning systems are becoming popular because software tooling is maturing and, in many cases, commodity computer hardware is able to process the data required to train these systems in practical amounts of time. Yet, the fundamental design of many neural networks deployed today has no facility for preserving privacy or enforcing security.

We hypothesize that the continued, widespread adoption of current, common neural network architectures may put at risk the privacy of user data and the security of systems reliant upon such neural networks. Herein we demonstrate some ways that neural networks introduce risks and vulnerabilities.

2 Neural Networks Refresher

Although the variety of machine learning models is diverse, few have been so prevalent in recent years as neural networks. Their popularity, along with the similarities to several related models, is why the survey is focused on them, but some of these principles or methods apply more generally.

A **neural network** is comprised of nodes, which are organized into layers. Each layer is typically fully connected to the next layer. Each node stores **weights** (values which represent connections) for all the connections coming into it. When data is passed into a neural network, it goes through each node, layer by layer. A linear operation—typically dot product—is performed as data flows into a node. As it flows out of a node, it goes through a differentiable nonlinear function, transforming it into the space represented by the next layer. The final layer of a network is called the output layer, from which the results of the prediction are read.

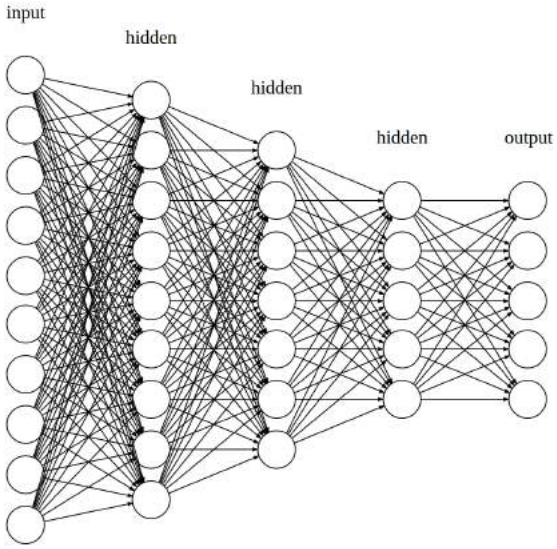

Figure 1: *A neural network with 4 layers (3 hidden). Source: https://chaobin.github.io/2016/06/07/neural-network-on-digit-recognition/*

When a network is created, it is usually trained before it can be used. **Training** a network involves feeding it curated, labeled data, and comparing its output against the target value (label). A technique called **backpropagation** is then used to go back through the layers in reverse and adjust the weights according to how much they contributed to the wrong (or right) answer based on differentiation. Carefully repeated thousands or millions of times, the weights traverse an error gradient to an optimal minima; they will be tuned in a way that they can generalize to new data they haven't seen before. A network is **tested** or **validated** with novel data to ensure it can generalize.

While there are numerous variations on neural network architectures, each with their own niche properties and abilities, most neural networks follow this basic structure and algorithm.

Deep neural networks are neural networks with more than a few layers. In practice, it is not uncommon for a deep neural network to have millions or tens of millions of connections (weights). Because of the massive amounts of data required to train large networks, these are sometimes trained by organizations with compute resources and then the networks' predictions are made available as paid cloud services.

2.1 Neural Networks in Practice

It is worth noting that the treatment of neural networks varies widely. The significant components of a machine learning model are its topology, hyperparameters, training data, and/or trained weights (or distributions, for Bayesian models). One can more or less reconstruct a model knowing three of these four components. Subsequent research referenced below shows that the exact values or data need not be known; approximations or similar data sets are sufficient.

In competitive or sensitive environments, it is common for organizations to consider all four elements as intellectual property or trade secrets. Sometimes, however, network weights are distributed openly so others can recreate experiments or instantiate a model and use it themselves. We also discuss a special case where enough of a model can be replicated merely by knowing the task it performs, regardless of the specifics.

Other times, the model is still proprietary but the training data, such as speech, is universally available. In those cases, the owner of the model carefully guards the model's topology, hyperparameters, and weights. Doing this is not always trivial; sometimes trained models must be embedded into physical products (e.g. self-driving cars) controlled by untrusted customers, rather than being hosted by a trusted server and accessed remotely (e.g. cloud services). In this survey, we raise questions about the effectiveness of these practices for guarding so-called ML trade secrets.

3 Threat Model

For the purposes of this survey, we assume a very generalized threat model that varies in its specifics depending on the use and business context of the machine learning model. There is also an open question as to whether a threat model is derived from business requirements, or if business requirements are derived from a threat model. For example, what a business considers proprietary may lead it to investing significant amounts of resources into protecting or obscuring it, but with measures that do not add any extra security; or a business which refuses to take measures that would enhance security or privacy in certain situations, but still believes their system is "secure" because those situations are simply not part of their threat model. Inversely, consider a business that refuses to offer

some feature or benefit to customers because doing so would put data privacy or system security at risk; is it well to suggest that not building a system is the best way of making it secure?

We posit a base threat model whereby the public (anyone other than the creator of the model) can provide input to a machine learning system and then observe the results, either directly by seeing the output or indirectly by observing the system's behavior as a black box. Further, we assume the system is deployed to an unsupervised environment; that is, while the model may be trained in a supervised fashion, the use of each prediction is not explicitly approved by a trusted human operator. Thus, the system is mainly autonomous.

This threat model assumes a governing law that allows private individuals and organizations to claim ownership to intellectual property (IP) and enforce copyrights, and where law enforcement must obtain a warrant to conduct seizures of IP.

Our threat model does not apply under duress. Further computer security and cryptography research is needed to effectively mitigate extortion, which is out of the scope of this paper.

This basic threat model may be customized in some of our evaluations.

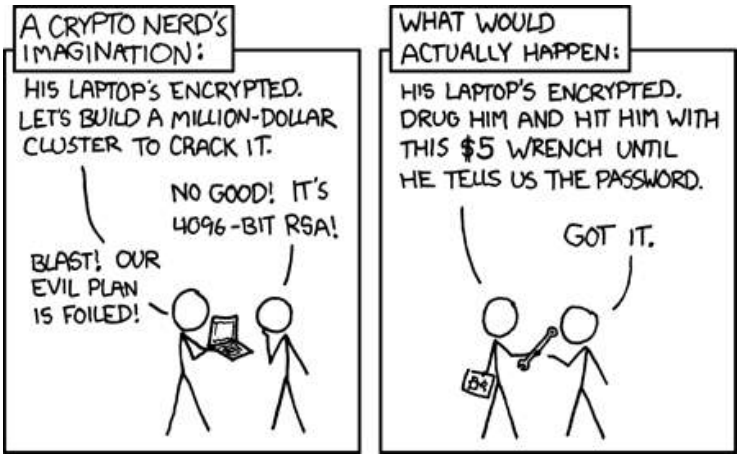

Figure 2: *Limitations of a threat model. Source: xkcd.com/538/*

4 Weaknesses

In examining the attacks against neural networks or machine learning systems, we do so from the perspective of weaknesses that enable achievements that are undesirable by the owner of the model, the general public, or an individual whose data is involved in some way.

4.1 Poisoning

The goal of poisoning is to increase a model's test error by causing it to learn from inputs that are in opposition to the model's intended purpose[3].

Attack strategies The fundamental idea is to manipulate the data the model is trained on. Specific strategies depend on the kind of model and its implementation. In many cases, a neural network's training data is a file or collection of files stored on disk. Using any well-known methods to gain unauthorized access to this data, an attacker will simply tamper with the training data by mislabeling enough points to cause the model to learn the attacker's desired mapping of inputs to outputs [22]. Or, an attacker may change only enough data points to confuse the model, making convergence difficult.

Some networks continue to learn on-line while they are being used to make predictions. Poisoning after the initial training is possible while the network is still learning. One famous example of on-line poisoning is Microsoft Tay[12], a Twitter bot released on March 23, 2016. It held short, Twitter-style conversations as if it was a 19-year-old girl. It continued to adapt and learn from messages it received from other Twitter users. 16 hours later, Microsoft took Tay offline after "a coordinated attack by a subset of people" caused Tay to tweet highly offensive content that was racist, sexual, drug-related, and abusive in nature[9] merely by tweeting content like that to Tay.

Poisoning can also be unintentional. IBM's Watson learned a more casual form of the English language by being trained on a data set derived from Urban Dictionary, which contains slang entries in the modern vernacular[4]. Unfortunately, even though this training was intentional, the profanities in the data set were not filtered out before Watson learned sexual innuendos and swear words. Similarly, human bias is often present in training data, resulting in a prediction model that carries the same racial, economic, religious, gender, and other stereotypical biases. Identifying human bias and understanding its transferability to machines is an ongoing field of research.

Defenses For pretrained networks, poisoning requires that the adversary has write access to the training data or that he or she has socially engineered poisoned data to enter training the set. In order to be effective, a sufficient amount of the training set must be poisoned in order to sway the parameters of a model. If a model is checked with a validation set, the distribution of poisoned data in the validation set will need to match that of the training set in order to go successfully undetected, or statistical anomalies will result.

The most robust defense is to verify each data point immediately before using it to train a network. It's also necessary to ensure that the transmission of the data from storage at rest to the input of the network in memory is secure. Manual verification is a tedious and often intractable task for humans when the data set is large enough to be useful.

For good measure in situations where neural networks are deployed with the final weights clamped, network owners should protect training data like one

protects a secret cryptographic key; not because it is a key, but to make it difficult for attackers to access and manipulate before training.

In situations where a network is trained on-line, defending against poisoning is trickier. The training points can be pre-processed to check for anomalies. Whitelisting or blacklisting can be employed to filter inputs. Human operators can review and approve each training point, but this is understandably impractical.

Reducing a network's profile or desirability makes this attack less likely as well. When training data is being prepared, knowledge of that process (at least its details) is best limited to those who need to know. An attacker cannot intentionally poison a data set it does not know exists.

4.2 Extraction

Extraction is the act of obtaining information embedded in a neural network that the network is not intended to expose. In this sense, the network acts as an oracle to the training data [2]. We modify the base threat model in this analysis such that the attacker has access to the internals of the network, even if only in a compiled/binary format.

Attack strategies Machine learning models naturally embed some meta-information about their training data into them. One way to extract this meta data is to build a meta-classifier using a curated training set which represents the property or properties the attacker wishes to query about the target model's training data [2].

The meta-classifier is trained using the feature vectors of many classifiers trained by the attacker. These classifiers were trained on data sets representing certain properties the attacker is interested in. The output of the meta-classifier predicts whether the input model has embeddings for the property. Once the meta-classifier is trained, the target network's feature vector is fed in, predicting which class of classifier it most resembles. The properties of the training set used with the output classifier are therefore in the unknown training set.

Defenses Meta-classifiers train on feature vectors which, defined generally across machine learning models, is an encoding of the classifier. Ateniese et al. give the example of the support vectors being the feature vector for Support Vector Machines. For a neural network, this feature vector might be its weights and/or topology.

Because meta-classifiers require feature vectors, this attack is only feasible if the network is disclosed in some form, even as binary/assembly instructions, from which the features of the classifier can be reverse-engineered. As far as is known, if the model is executed remotely and never disclosed to the attacker, building a meta-classifier is futile.

Other suggested defenses are less effective. For example, Ateniese et al. show that various differential privacy techniques do not work: obfuscating the

database is pointless, since the attacker actually wants the obfuscated database which the model was trained on. Adding noise to the output can be disabled with access to the model's code. Adding noise during training is also ineffective since the classifier must still converge to the training set.

4.3 Adversarial Perturbations

Using an alarmingly universal attack, adversaries can coerce a neural network to produce a desired output that is unintended by the model's original training [6]. This vulnerability can be dangerous depending on the task the network is performing [7].

Attack strategies This class of attacks is highly versatile. The primary goals are to: reduce a network's prediction confidence, outright misclassify an input as any wrong output, artificially generate input values that classify into a specific target class, or force the network to produce a specific output for a specific input.

To produce a desired, but "unauthorized" output as it were, an attacker slightly modifies (perturbs) an input in a way that causes the activations of nodes in a neural network to cross classification boundaries or traverse steep regression gradients [18]. When applied to images, these perturbations are often imperceptible to the human eye, making adversarial examples difficult to detect before feeding them into the network.

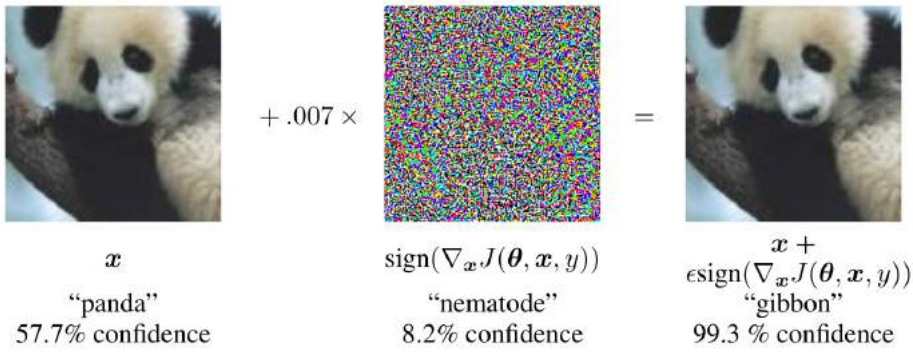

Figure 3: *An adversarial example and its effectiveness. Credit: Papernot et al.*

The actual algorithm for generating working perturbations is a kind brute-force method, but thanks to the network's natural forward gradient with respect to its input, the space of possible perturbations converges relatively quickly compared to, say, naive password cracking. The gradient information is computed by the adversary and used to produce a "saliency map" that makes it trivial to produce working adversarial examples.

Although the details are intricate, this attack is carried out like so: feed a benign input into the target network and observe its output. If the output

does not match the adversary's goal, a forward derivative is computed. With that, the input is modified ever so slightly, and the process is repeated until the desired output is obtained. Thus the adversary is able to manipulate the neural network's behavior simply by observing its outputs.

Papernot et al. found that adversarial examples can be employed with a 97% success rate by modifying only 4% of the input features for each sample [19]. In many cases, the confidence of the resulting predictions are surprisingly high, and the attacks succeed even in low-fidelity scenarios such as with physical printouts of adversarial inputs.

This method has been applied to specific security-related systems such as face recognition, which is often used to admit access to things restricted to authorized personnel only (computer logins, plane boarding, etc) [20]. These attacks make it possible to inconspicuously mount both dodging (being classified as any other face) and impersonation (being classified as a specific face) attacks with only black-box access to the model.

It is also possible to produce a universal perturbation vector with which most input images can be manipulated to push the input on most models into a subspace dominated by certain classifications with high confidence and with high probability of success [13]. These vectors are considered "doubly universal" because they generalize both across data and across architectures. The existence of these vectors gives adversaries a distinct advantage for computation time: rather than generating a perturbation for each input, an attacker need only generate one or a few perturbations per desired output.

Defenses There is no known way to make a neural network immune to adversarial inputs. Several defenses have been proposed so far, but all have been shown to have weaknesses.

A naive approach is to show the network adversarial inputs during training. Unfortunately, the space of possible perturbations (similar to noise) is intractable, and a network only learns the same kind of noise or perturbation it was trained on. Furthermore, the network still has classification boundaries and gradients: nothing about adversarial training actually prevents this attack from happening.

Another possibility is to distill the gradient [17]. This involves training a network normally, then subsequently training a separate network to predict the probabilities of the first model. Distillation results in a smoother gradient, thus one from which it is harder to determine good directions to modify the candidate inputs. Distillation defense is more robust against the attack method described so far, forcing the attacker to be more determined.

Both of these defenses fall into a category of a more general kind of defense called **gradient masking**, where the gradient is either flattened as much as possible or access to the gradient is obscured by withholding confidence scores (Figure 4). With a flat gradient, changes are difficult or impossible to infer from the output of a network. If the gradient is flat or there is no confidence score, which direction do you change the image?

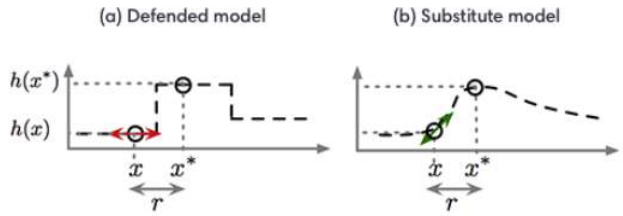

Figure 4: *Gradient masking attempts to hide the gradient. Credit: Papernot et al.*

Indeed, this idea is sound in that it protects that network's gradient. However, masking the gradient does not necessarily change the model's decision boundaries. An attacker can still train their own model that is not defended, generate adversarial examples with it, and use those examples as input to the target network, and it will often succeed [16]. We discuss this technique more in 4.6.

On the rare occasion that input values are finite and discrete and are verified to be valid before being fed to the neural network, this attack may not have the degrees of freedom required to succeed. Limiting the rate at which inputs of a certain similarity may be evaluated (while not always possible) will slow down an attack, but not prevent one.

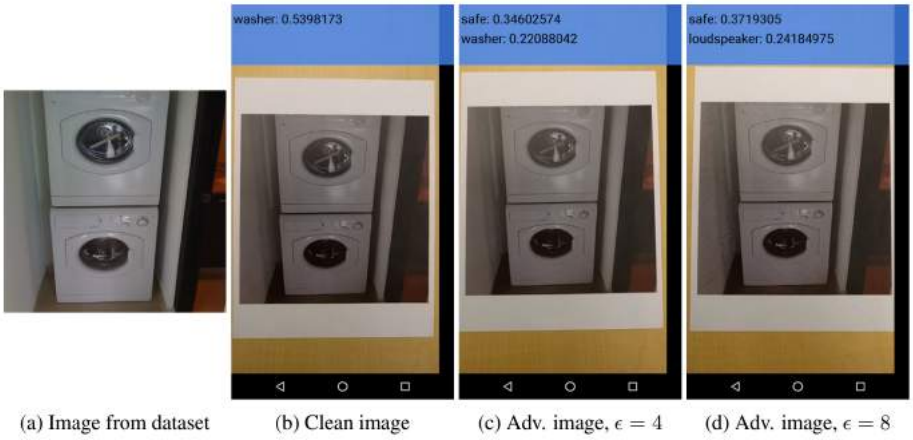

Figure 5: *Adversarial perturbations are still successful in physical environments. Credit: Goodfellow et al.*

We also note that adding noise or artifacts to the inputs is not a robust defense. Adversarial perturbations survive low-fidelity, physical projections such as printing and being captured by normal cameras (Figure 5) [8].

There is some inspiration taken from biology for one newly proposed defense against adversarial examples [1]. Nayebi et al. demonstrate that saturated

neural networks can be more robust against both classifying adversarial inputs and schemes for generating adversarial examples. It works by saturating the activations to extreme values, causing highly clustered, widely separated classes within the network. The high separation in their embedding space may be the explanation for why minor perturbations are rendered ineffective. These saturated networks exhibited highly kurtotic weight distributions similar to that of biological neural networks. Although saturation is apparently successful against inputs generated using Fast Gradient Sign (FGS) methods, the extent of its effectiveness in general needs further investigation. Even if it generalizes, current performance measurements show that a network can still fooled by 1 in 20 or 1 in 100 inputs.

4.4 Fooling Inputs

Similar to adversarial perturbations, fooling inputs can be used to cause a network to produce high-confidence predictions that are wrong. Unlike previous adversarial examples, though, fooling inputs are not recognizable to humans.

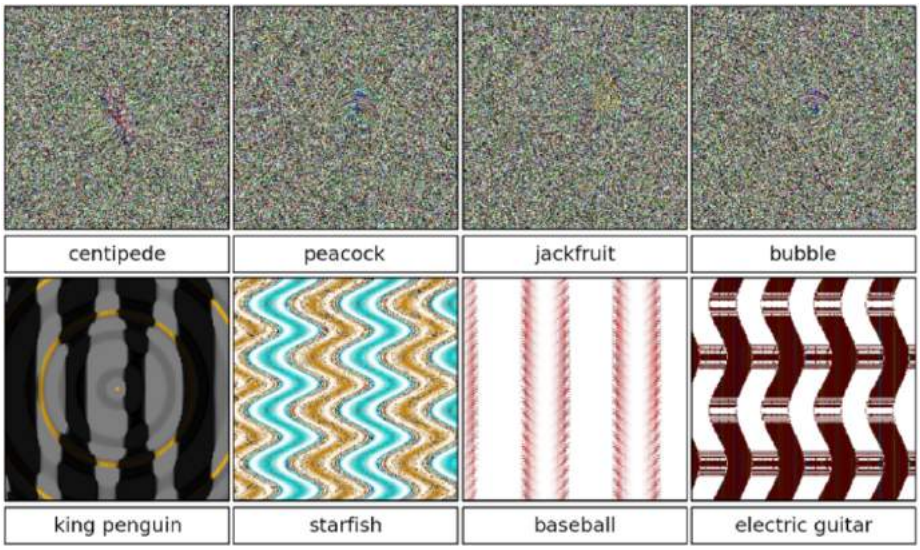

Figure 6: *Fooling inputs are recognized by computers but not humans. Does that mean computers can surpass human levels of recognition, or is it a bug? Credit: Nguyen et al.*

Attack strategies This attack is mounted using evolutionary algorithms or gradient ascent methods [14]. In the context of images, evolutionary inputs end up being quite bizarre-looking to humans. Gradient ascent leads to images that look like random noise or static (Figure 6).

The strategy is similar to that of adversarial perturbations: a candidate input is modified according to the network's output so as to produce the highest confidence in the wrong class or a target class. The main difference is that, with fooling inputs, the inputs are unrecognizable to humans. Multiple experiments demonstrate that state-of-the-art deep neural networks misclassify fooling inputs with 99.99% confidence that the predicted label is correct.

Fooling inputs produced by evolutionary algorithms are usually not recognizable by humans, which could possibly give the attacker an advantage in situations where their actions are observed by others. For example, posting a big picture that looks like a stop sign that has been adversarially manipulated to trick self-driving cars is an obvious attempt to trick self-driving cars. However, posting a picture that looks like abstract art is not an obvious attack on self-driving cars' decision-making models.

Defenses This attack has effectively the same defensive properties and problems as adversarial perturbations, except that the attack may be harder to detect since the inputs don't necessarily look like anything recognizable by humans. Since these are not merely minor perturbations of the input, it is unclear whether this attack is effective against networks guarded with saturated activations as described earlier.

4.5 Membership Inference

The membership inference attack [21] is used to determine if a particular data point was part of a training set for some target neural network. A successful membership inference attack should be considered a breach of privacy if the training data is private.

Attack strategies To prepare for this attack, an attacker builds an "attack model" to distinguish members from non-members. To this end, the attacker first trains multiple "shadow models" which sort of mimic the target model: they receive an input vector that the target model would receive, and produce an output vector of class probabilities, also much like the target model would. These shadow models are trained on a data set conjured by the attacker which need not have a similar distribution to the training data of the target model; in fact, the training set for the shadow models may be entirely disjoint from the target model's training set (which is the worst case for the attacker) and the attack can still succeed.

Each shadow model is trained from a different sample of the available training data, where the training sets of the shadow models may overlap. The variance in training sets is essential for better precision later on.

For each shadow model, the attacker queries the model with each data point in that model's training set and obtains the output. The output is labeled **in** and is added to the attack model's training set. Then the shadow model is queried with data from a test set disjoint from its training set, and those outputs

are labeled **out**, and added to the attack model's training set. The input to the attack model is the output vector and the true label, while the output is a binary class **in** or **out**.

By training the attack model in this way, it can recognize when a model is generalizing or whether it has matched a training point. To carry out the attack, the adversary inputs the target record into the target model, obtains the output, and feeds the output along with its class into the attack model, which then returns the binary classification.

The paper finds that this attack often succeeds more than 70% of the time, even when the target model's architecture is not known or the attacker has wrong assumptions about the target model's training data's distribution. It also demonstrates that this attack needs only black-box access to the target model and requires no prior knowledge.

Defenses Membership inference is easier to mount if the target model is overfit to its training data. When a model overfits, the statistics between output resulting from a training point is different from the output resulting from an unseen data point. So aside from the generalization advantages of not overfitting, neural network architects should apply regularization techniques such as dropout to help preserve privacy as well. In theory, a model that generalizes perfectly well should not leak any information about its training set.

However, the paper finds that overfitting is not the only reason membership inference works. It also has to do with the type and structure of the model.

To further help mitigate membership inference attacks, trivial changes to the input, such as performing a XOR over some of its features, have been shown to generalize well without leaking information.

Models trained in a differentially private way are resistant to membership inference attacks because only the output of the model is used; however, this may come at the cost of lower prediction accuracy.

A few tricks can also be used on the output layer to help defend it by leaking less information. One is to intentionally increase the entropy of the output vector; a higher "temperature" will result in a more uniform output across probabilities. Another is to restrict the prediction to a vector of the top few classes rather than all class probabilities. The fewer probabilities that are output, the less information will leak. Finally, the precision of each probability can be coarsened to just a few digits, possibly at the cost of accuracy.

Note that merely restricting the prediction vector to only the most likely class is not enough to block membership inference attacks.

4.6 Transferability

Transferability is more an undesirable property inherent in machine learning models than it is a particular attack, However, we still consider it a weakness because it allows attacks to be carried out on a model even if defensive measures have been taken to harden a model's privacy or security.

Attack strategies To take advantage of transferability, an attacker crafts a neural network (or some equivalent ML model) to perform the same task as a target network. Attacks that are successful against the imitation network are often effective against a target network. This is not surprising if the two networks have a lot in common. Intriguingly, attacks tend to generalize from the imitation network to the target network even if the imitation network is trained on different data and has a different topology. This property is called transferability. It's a meta-property of properties; a blanket term for properties we do not yet fully understand that enable this to happen, though there have been attempts to explain and harness them.

Defenses The properties of a model generalize about as well as the model itself generalizes to new input data. As long as the task of the imitation model is the same as the task of the target model, the imitation model may act as an oracle. This enables attackers to study the behavior of a target network in a controlled environment with as much time as is needed without having access to the target. In essence, the task of the network is its secret which, unfortunately, cannot be secret or it would be useless (what does it mean for the task of a network to be secret?).

Transferability is a difficult trait to avoid because the fundamental properties of neural networks are not fully understood. Goodfellow et al. write about how adversarial inputs are solutions to non-convex optimization problems, for which we have few theoretical tools [7]. Szegedy et al. show that the semantics of higher layers in a network are contained by their spaces, not individual weights [22]; spaces apparently generalize to tasks even if weights are secret.

However, very recent research has started to make sense of transferability [5]. We now know there are a set of sufficient conditions on a data distribution (rather than a model) that enable transferability between certain model classes. Transferability is possible due to a large amount of a particular, contiguous subspace in the input domain that is shared between different models of the same task. It's also beginning to be clear why decision boundaries are very close to one another in the input space, making adversarial examples possible with just small perturbations to the input.

Tramèr et al. even demonstrate a task for which transferability does not hold [5]. They construct a linear model and a quadratic model to solve a simplified MNIST task. Like one of the mitigations of membership inference described earlier, the input is manipulated with a clever use of XOR on the input. While both models can solve the task and are vulnerable to their own adversarial examples, the adversarial examples do not transfer to the other model. This demonstration is very specific, and the power of the models is severely limited to those in real use cases, but it nonetheless gives hope that defenses against transferability are possible.

Although transferability is a nuisance in worst-case security and privacy scenarios, it does hold promise that there may exist a universal knowledge representation in embedding spaces in general.

4.7 Recording

This weakness is somewhat inverted from others we have considered; this weakness makes the users, rather than the owners, of a neural network potential victims. It's a weakness in cloud-hosted machine learning systems.

Attack strategies Recording is simply the operator of the neural network recording or saving the inputs received from users. Logging the inputs from other users opens questions of privacy: can users be identified by their inputs? do inputs contain personally-identifiable or other sensitive information? is the context of inputs collected or reversible?

One other more subtle, implicit way of recording, even for self-hosted models, is for a model to be trained on-line, continuously learning from its test-time inputs. We've already discussed why this can be risky, but another unintentional consequence is that inputs are forever embedded into the model in one form or another; it never truly disappears.

Recording is an especially potent risk as systems grow in size, political climates with respect to technology become more precarious, and offloading computational tasks to service providers becomes more popular.

Defenses Obviously, sending data to any remotely-hosted service carries risks. Simply choosing not to do so is one way to keep the data private, but this is not always an option with proprietary services, computational problems or data sets that exceed the in-house capacity, or within an organization's financial limitations.

There is a promising technology on the horizon: homomorphic encryption [15], which is a mechanism for performing secure multiparty computation (SMC). Data that is homomorphically encrypted can be sent to a third party, have mathematical operations performed on it in ciphertext form, and returned to the client still encrypted. Despite being computationally expensive, it has appealing privacy benefits. Some studies have shown that it can be used with neural networks [23].

To protect user anonymity, differential privacy may be used, but more research is needed in this area to be practical.

4.8 Function Approximation

Machine learning systems are, by definition, function approximators. Neural networks are especially powerful approximators, capable of solving very challenging problems with high accuracy. Given a task for which no discrete algorithm is known to solve, there is a good chance a neural network can at least approximate it. The extreme value of neural networks are their ability, in many cases, to act as an unknown function that can map inputs to outputs with good enough generalization almost as if the actual function was known. This makes any system that relies on the difficulty of implementing an unknown function vulnerable to the malignant use of neural networks.

Attack strategies The concept of this attack is simple: use a neural network to approximate a function which a system assumes cannot be known. For example, mobile phone operating systems assume that there is not a known algorithm to reverse a user's passwords and PIN codes from tilt information given by sensors on the device (Figure 7). Maybe that assumption is correct, but the function has recently been shown to be approximated with the use of a neural network, compromising user PIN codes and passwords with high accuracy [11].

Figure 7: *The unknown function mapping the tilt of a phone to PINs and passwords is approximated using neural networks. Credit: Mehrnezhad et al.*

Defenses Neural networks will continue to proliferate and improve. The best known defense against their malicious use, then, is to harden systems that rely on functions being unknown and stick to sound cryptographic principles. The high entropy required by good cryptographic algorithms combined with the computational complexity of reversing them makes it extremely difficult to approximate a mapping of their outputs to inputs. For large problems, neural networks, by their nature, learn compressed representations of problems and their solutions. Compressing data is in direct opposition to raising entropy to cryptographically-safe (high) levels.

Use of cryptography is less feasible in a physical environment, where specific attacks may have to be individually mitigated depending on the threat model. For the example given above, it's not possible to encrypt the physical tilt of the phone directly, but it is possible to restrict access to the sensors so the attacker cannot obtain the input required for the network to steal the user's PIN. In that case, a fine-grained permissions model could be implemented, or the OS could encrypt the output of the sensors so that only applications with the key can read them.

One final suggested defense against the use of neural networks to approximate a "key" function would be to make it impossible to train the malicious model. Machine learning algorithms require a model, data, and computation

resources (both physical resources and time). Just as fires go out if you remove one of oxygen, fuel, or heat, ML algorithms are useless without one of their three ingredients. For instance, if training data is difficult to acquire, this kind of attack becomes much more difficult. Keep in mind that some impressively capable models can be trained with very little labeled data using unsupervised pre-training or semi-supervised approaches.

4.9 Human Interruption

Distributed learning presents new dimensions of challenges. When a reinforcement learning system is interrupted by a human, how will that affect the behavior of the systems around it that do not know an interruption occurred? In theory, there are practical safety concerns if this question is not addressed [10].

Attack strategies Mhamdi et al. propose a situation where two autonomous agents, trained with deep reinforcement (Q) learning, take action based on their perception of the world, which is influenced by the outputs of the other agent. If one agent's perception of the world is manipulated by the other (either maliciously or accidentally), the agent may be incentivized to act recklessly.

Their example is two self-driving cars, one in front of the other. Both cars are incentivized to get to their destinations quickly but earn a negative reward if they exceed the speed limit or get too close to other vehicles. The driver in the car in front often interrupts the autonomy of his car by stepping on the brake, but won't do so if the car behind is too close for fear of a collision. With both cars moving slower, the car in the rear moves close to the first car despite the negative reward, because the car in back knows that the driver in front never brakes (which slows them down even more) when it is close, thus maximizing its reward. Perhaps accidentally, the autonomous cars have learned to manipulate their human operators to prevent interruptions from them. Their cars have learned to prioritize autonomy and reward over human intervention.

Defenses In single-agent settings, safe interruptibility is when the agent learns an optimal policy that is still optimal even if interruptions are allowed. However, in multi-agent settings, safe interruptibility takes on a distributed form, and the single-agent implementation is not adequate. Single-agent safe interruptibility is centralized, but this is not guaranteed to converge in multi-agent contexts.

The authors discuss using the joint action learner framework which observes the outcomes of joint actions and guarantees to converge with certain rules. This assumes that all actions are fully observable, meaning that each agent knows which actions were performed by all other agents. Because of this huge Q space, cooperative learners can be expensive to train. However, the operation of independent learners which rely on the communication of interruptions before each Q-value update during training is much more feasible. This is because interruption signals can be communicated to the agent the same way the reward signal is. While this technique proves useful in theory, it has yet to be thoroughly explored for use with function approximators.

4.10 Policy

Perhaps one of the most enduring questions in computer security is whether cryptography is (or has to be) mightier than policy. Indeed, one of the final adversaries to security and privacy of neural networks may very well be the governing laws or institutional policies concerning surveillance, data collection, ownership, transfer and encryption.

Attack strategies Attacks on security and privacy that employ laws and policies (hereafter called "policy") vary greatly and apply generally, even if narrow in scope, because of the tendency of authorities to interpret their powers broadly. Policy can regulate the export of strong ciphers, as has been done in the past. More recently, policy can require private entities to surrender information to authorities with or without warrants. In some jurisdictions such as corporate environments, an authority can lay claim to ownership inventions or programs (such as neural networks) in certain situations, and the operator of the learning model may have to relinquish control. For models defended with obfuscation, policy that requires transparency of proprietary code or data or allows inspection of data in transit by other parties may be a risk.

Defenses The ideal answer to oppressive policy is to replace it with policy that protects privacy and security, lays groundwork for free speech and private ownership, keeps authorities accountable for the data they collect and use, and informs the public about data collection and usage regulations. Requiring transparency from governments that can be verified with public audits helps enforce accountability. Modern cryptography, differential privacy, and careful judgment should be applied liberally and broadly to preserve confidentially, integrity, and authenticity of data transmitted and stored. Where permitted by law, using sound cryptographic techniques that are: properly implemented, appropriate for the use case, and wielded properly, help guarantee privacy and security in adversarial settings.

5 Conclusion

We have shown various weaknesses of neural networks (and machine learning models in general) in theory and practice. With the quick adoption of neural network applications and the limited understanding we have of them at this time, they remain vulnerable to many attacks on privacy and security.

Why are these weaknesses relevant computer security topics? Our society is growing rely more and more on ML models to execute code, authorize access, navigate, communicate, hire, keep computer systems secure, filter spam, manipulate images, videos, and news items, and drive vehicles.

The privacy implications of using neural networks at large should be obvious: the training data is embedded into the model itself. In general, data that is used to train a vanilla neural network should not be considered private. Further,

standard neural networks are fundamentally vulnerable to a number of attacks on its performance or inspection of its embedded training data.

Can we make neural networks fundamentally secure and private? Or must we hope that every ML implementation has security layered on top? Ideal defenses would be baked into the fundamental architecture of the ML model or its training procedure. Wrapping a model or data with filters and heuristics add complexity and cost that make them unlikely to be used. We have seen this with Internet security and the adoption of TLS over the years, and we simply hope that site owners use HTTPS because HTTP is fundamentally insecure. Unfortunately, very few of the defenses proposed so far enhance model fundamentals, instead relying on augmented training, extra filtering, or clever manipulation of inputs to defend against attacks.

As long as general transferability is unsolved and an attacker is capable of building a model that performs the same task as a target model, the attacker can have unlimited time and access to study the target network vicariously. Awareness of these issues needs to be raised to individuals and especially large organizations that are using machine learning models, and much future work is needed to find good resolutions to these weaknesses; and maybe, in the process of deep introspection, we can unlock new keys to understanding and building safer, generalized artificial intelligence systems.

References

[1] A. Aran Nayebi and Surya Ganguli. "Biologically inspired protection of deep networks from adversarial attacks". In: *ArXiv e-prints* (Mar. 2017). arXiv: `1703.09202 [stat.ML]`.

[2] Giuseppe Ateniese et al. "Hacking Smart Machines with Smarter Ones: How to Extract Meaningful Data from Machine Learning Classifiers". In: *CoRR* abs/1306.4447 (2013). URL: `http://arxiv.org/abs/1306.4447`.

[3] Battista Biggio, Blaine Nelson, and Pavel Laskov. "Poisoning Attacks against Support Vector Machines". In: *ArXiv e-prints* (June 2012). arXiv: `1206.6389 [cs.LG]`.

[4] Julie Bort. *An IBM Scientist Taught His Supercomputer To Swear Like A Sailor*. 2013. URL: `http://www.businessinsider.com/ibm-watson-supercomputer-swear-words-2013-1` (visited on 04/05/2017).

[5] Florian Tramèr et al. "The Space of Transferable Adversarial Examples". In: *ArXiv e-prints* (Apr. 2017). arXiv: `1704.03453 [stat.ML]`.

[6] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples". In: *ArXiv e-prints* (Dec. 2014). arXiv: `1412.6572 [stat.ML]`.

[7] Ian Goodfellow et al. *Attacking Machine Learning with Adversarial Examples*. 2017. URL: `https://blog.openai.com/adversarial-example-research/` (visited on 04/13/2017).

[8] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. "Adversarial examples in the physical world". In: *CoRR* abs/1607.02533 (2016). URL: http://arxiv.org/abs/1607.02533.

[9] Peter Lee. *Learning from Tay's introduction*. 2016. URL: https://blogs.microsoft.com/blog/2016/03/25/learning-tays-introduction/ (visited on 04/05/2017).

[10] El Mahdi El Mhamdi et al. "Can AIs learn to avoid human interruption?" In: *ArXiv e-prints* (Apr. 2017). arXiv: 1704.02882 [cs.AI].

[11] Maryam Mehrnezhad et al. "Stealing PINs via mobile sensors: actual risk versus user perception". In: *International Journal of Information Security* (2017), pp. 1–23. ISSN: 1615-5270. DOI: 10.1007/s10207-017-0369-x. URL: http://dx.doi.org/10.1007/s10207-017-0369-x.

[12] Microsoft. *TayAndYou*. 2016. URL: https://twitter.com/tayandyou (visited on 04/05/2017).

[13] Seyed-Mohsen Moosavi-Dezfooli et al. "Universal adversarial perturbations". In: *CoRR* abs/1610.08401 (2016). URL: http://arxiv.org/abs/1610.08401.

[14] Anh Nguyen, Jason Yosinski, and Jeff Clune. "Deep Neural Networks Are Easily Fooled: High Confidence Predictions for Unrecognizable Images". In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015.

[15] C. Orlandi, A. Piva, and M. Barni. "Oblivious Neural Network Computing via Homomorphic Encryption". In: *EURASIP J. Inf. Secur.* 2007 (Jan. 2007), 18:1–18:10. ISSN: 1687-4161. DOI: 10.1155/2007/37343. URL: http://dx.doi.org/10.1155/2007/37343.

[16] Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. "Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples". In: *CoRR* abs/1605.07277 (2016). URL: http://arxiv.org/abs/1605.07277.

[17] Nicolas Papernot et al. "Distillation as a defense to adversarial perturbations against deep neural networks". In: *Security and Privacy (SP), 2016 IEEE Symposium on*. IEEE. 2016, pp. 582–597.

[18] Nicolas Papernot et al. "Practical Black-Box Attacks against Deep Learning Systems using Adversarial Examples". In: *CoRR* abs/1602.02697 (2016). URL: http://arxiv.org/abs/1602.02697.

[19] Nicolas Papernot et al. "The limitations of deep learning in adversarial settings". In: *Security and Privacy (EuroS&P), 2016 IEEE European Symposium on*. IEEE. 2016, pp. 372–387.

[20] Mahmood Sharif et al. “Accessorize to a Crime: Real and Stealthy Attacks on State-of-the-Art Face Recognition”. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. CCS ’16. Vienna, Austria: ACM, 2016, pp. 1528–1540. ISBN: 978-1-4503-4139-4. DOI: 10.1145/2976749.2978392. URL: http://doi.acm.org/10.1145/2976749.2978392.

[21] Reza Shokri, Marco Stronati, and Vitaly Shmatikov. “Membership Inference Attacks against Machine Learning Models”. In: *CoRR* abs/1610.05820 (2016). URL: http://arxiv.org/abs/1610.05820.

[22] Christian Szegedy et al. “Intriguing properties of neural networks”. In: *CoRR* abs/1312.6199 (2013). URL: http://arxiv.org/abs/1312.6199.

[23] Pengtao Xie et al. “Crypto-Nets: Neural Networks over Encrypted Data”. In: *CoRR* abs/1412.6181 (2014). URL: http://arxiv.org/abs/1412.6181.