Machine Learning Course - CS-433

Adversarial ML

Dec 5, 2019

Small changes by Martin Jaggi 2019;

Last updated on: December 5, 2019

Introduction

Some ML tasks are inherently difficult. E.g., consider the examples in Figure 1. Even humans might not be able to classify all examples correctly. So we would not be surprised to see NNs struggle in such cases. But typically this is where they shine, having a performance on par or perhaps even surpassing humans. But to date, NNs (or other classifiers for that matter) are not as *robust* in their decisions as humans and can be easily tricked by adversarially chosen perturbations of the input, even if the perturbations are small. This can lead to problems.

Figure 1: Dog or mop?

In the sequel it might be good to have a concrete example in mind. E.g., think of self-driving cars. And we will assume that we are using NNs. But the basic principle applies to a much wider setting.

So consider a self-driving car. Such a device hopefully will be able to recognize and correctly interpret street signs with very high probability. If the ML algorithm that performs

this task can be easily tricked by slightly manipulating the input then insurance companies might not be happy![1]

The Basic Attack using Back Propagation

So let us look at the simplest instance. Assume that we have trained a binary classifier $f : \mathcal{X} \to \{\pm 1\}$ given some training set $\mathcal{S} = \{(\mathbf{x}_n, y_n)\}_{n=1}^N$. There is an underlying data distribution $\mathcal{D}$ that is unknown to us. Assume that we have trained the network well and that it has a small true risk, i.e.,

$$\mathcal{L}(f) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}[\mathbb{1}_{\{f(\mathbf{x})\neq y\}}] \leq \delta,$$

where $\delta > 0$ is a small number. We are happy.
But assume now that an adversary were allowed to manipulate, i.e., change, the input $\mathbf{x}$ slightly. How much worse can the risk be made? If we put no restriction on the "power" of the adversary then clearly she can increase the risk at will. So let us say that the adversary can change the given input $\mathbf{x}$ to $\tilde{\mathbf{x}}$, but that we require that $\|\mathbf{x} - \tilde{\mathbf{x}}\| \leq \varepsilon$ for some small $\varepsilon > 0$. What norm shall we pick? This depends on the

[1] Just to clear up any confusion. There are also GANs – which confusingly stands for generative adversarial networks. Even though there is also the name "adversarial" in this topic, this has nothing to do with the current set-up. In GANs the adversarial view-point helps to train a network. E.g., assume that the task of the network is to to create realistically looking faces given "random-like" input. Here the network is used in a "generative" way. It has at it's disposal a set of human faces. In order to train this network to perform its task better we use a second network that tries to distinguish between faces generated by the network and real faces. This is the adversary. Training now proceeds in two phases that are interlaced. Given a generative network we train a powerful adversary to perform the distinction and given an adversary we train a hopefull better generative network.

application and it is part of what is called the *threat model*. We will get back to this point later. It is customary in the literature to pick either ℓ_1, ℓ_2, or ℓ_∞. We can now define the adversarial risk, call it $\mathcal{R}(f, \varepsilon)$, as

$$\mathcal{R}(f, \varepsilon) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\Big[\max_{\tilde{\mathbf{x}}:\|\mathbf{x}-\tilde{\mathbf{x}}\|\leq\varepsilon} \mathbb{1}_{\{f(\tilde{\mathbf{x}})\neq y\}}\Big].$$

In words, for every input $\mathbf{x}$ the adversary can find the worst perturbation allowed by the norm constraint.
Depending on whether you are interested in "breaking" the classifier or try to make it robust we are faced with numerous questions. Here are some in no particular order.

1. How do we find adversarial perturbations efficiently?
2. In order to find those perturbations, what access to the classifier do we need?
3. By how much worse can we make the risk?
4. Are there measures we can take to make a given classifier more robust?
5. Are there particular ways to train the classifier to make it robust?

We will explore some of these questions in the following sections.

White Box Attacks

So assume that we are given a binary classifier f. To be concrete let us assume that it is implemented by a NN. We

have complete access to the NN and are given an input $\mathbf{x}$. How do we find an adversarial perturbation $\tilde{\mathbf{x}}$ so that $\|\mathbf{x} - \tilde{\mathbf{x}}\| \leq \varepsilon$? You will explore this in the exercise session. Here is a the basic idea.

If f does not classify $\mathbf{x}$ correctly then we are done – we can just set $\tilde{\mathbf{x}} = \mathbf{x}$. So we might as well assume that it does indeed classify $\mathbf{x}$ correctly. To be specific, assume that the data is separable in principle, i.e., that there exists a "ground truth" encoded by the function $h : \mathcal{X} \to \{\pm 1\}$. In other words, the correct label is non-probabilistic and is given by $h(\mathbf{x})$. Further assume that $f(\mathbf{x})$ has the form

$$f(\mathbf{x}) = \begin{cases} 1, & 0 \leq \frac{1}{2} \leq g(\mathbf{x}) \leq 1, \\ -1, & 0 \leq g(\mathbf{x}) < \frac{1}{2}, \end{cases} \tag{1}$$

where $g : \mathcal{X} \to [0, 1]$ represents the probability that $y = 1$ given the input. I.e., we assume that, like in logistic regression, we first compute a probability and then we quantize.

Of course $g(\mathbf{x})$ depends on all the weights and bias terms within the NN but we omit this dependence in our notation since in the present context we are interested in variations dues to changes in the input rather than changes in the parameters.

Why do we assume that f has this form? We will use gradient descent in order to find adversarial perturbations. For this to work, we want the objective function to be smooth. We will therefore use g rather than the original f.

Using the backpropagation algorithm, we can efficiently compute $\nabla_{\mathbf{x}} g(\mathbf{x})$. Note that this is the gradient with respect to changes in the input, rather than changes in the parameters.

Hence

$$h(\mathbf{x})\nabla_{\mathbf{x}} g(\mathbf{x})$$

is a vector of length D (the dimension of the input space) which is positive in positions where an increase in that input makes the prediction more correct (increases the probability of the correct label) and negative where an increase in that dimension makes the prediction less correct.

Assume that we are allowed to move the point $\mathbf{x}$ to a new point $\tilde{\mathbf{x}}$, with the restriction that $\|\tilde{\mathbf{x}} - \mathbf{x}\|_2 \leq \varepsilon$. Note that we have assumed an ℓ_2 constraint. Our aim is to make the prediction as bad as possible. This means that we would like to decrease the probability of the correct label as much as possible.

If ε is small then it makes sense to assume that the change in the probability is given by the first order changes, i.e., that it is well predicted by the gradient. Given our "budget" in terms of ℓ_2 the optimum move is then to move in the opposite direction of the gradient, i.e., to define[2]

$$\tilde{\mathbf{x}} = \mathbf{x} - \varepsilon h(\mathbf{x}) \frac{\nabla_{\mathbf{x}} g(\mathbf{x})}{\|\nabla_{\mathbf{x}} g(\mathbf{x})\|_2}.$$

If, for a particular $\mathbf{x}$, we manage to "flip" the probability using a move that are bounded by ε then we have found an adversarial example. In general, we might not want to take a single step but rather only move partially in this direction and then iterate this process.

[2]Mathematically speaking this corresponds to the following. We are given a fixed vector $\mathbf{w}$ of unit norm (in the ℓ_2 sense). Then the inner product $\mathbf{w}^\top \mathbf{v}$ conditioned on $\|\mathbf{w}\|_2 \leq \varepsilon$ is maximized by chosing $\mathbf{v} = \varepsilon \mathbf{w}$. This can be seen e.g. by the Cauchy Schwartz inequality $\mathbf{w}^\top \mathbf{v} \leq \|\mathbf{w}\|_2 \|\mathbf{v}\|_2$.

The above attack is known as a "white box" attack – we have access to the details of the algorithm.
Here is a good problem to think about. Assume that we are given as above the gradient $\nabla_{\mathbf{x}} g(\mathbf{x})$. What is the locally optimal move if our constraint is $\|\tilde{\mathbf{x}}-\mathbf{x}\|_1 \leq \varepsilon$ or $\|\tilde{\mathbf{x}}-\mathbf{x}\|_\infty \leq \varepsilon$ instead of the ℓ_2 constraint we used above?

Black Box Attacks

In the above attack we needed access to the NN that implemented the prediction in order to compute the gradients. In general it is a good idea in anything involving security to assume that the adversary has this level of access. Assume e.g., autonomous cars. Typically those are sold in large quantities and it will not be difficult for an adversary to get a hold of one of those boxes.
But even if we limit the adversary it turns out that similar attacks can still be carried out. Those are typically known as "black box" attacks. In such attacks we assume that we can observe the input out relationship but we cannot look inside the box (hence "black box"). Early on in the development of adversarial machine learning it was assumed that attacks could be prevented by preventing access to the algorithm or by obfuscating or masking the gradients (e.g., by adding some small amount of noise to the output or other similar techniques). But it has been shown that all such approaches can be easily broken and that black box access is enough. Either, one can compute gradients numerically by repeatedly querying the box or one can perform the following more interesting approach which hints at some "universality"

of many of the found adversarial directions. This second approach is the following.
Given the black box, create a new sample $\mathcal{S}$ by computing many input-output pairs. Given $\mathcal{S}$ train a new NN. Perhaps surprisingly it does not have to have the same structure (depth, width, activation functions etc) as the original one. Now run a white box attack on the newly-trained NN. Those adversarial direction that are found in this way often also cause trouble for the original network. This is quite interesting and non-trivial!

Adversarial Attacks on Physical Objects

So far in our discussion we have assumed that we are given an input $\mathbf{x}$ and that we are finding an adversarial perturbation. But there exist much more interesting and potentially more dangerous variants. Let us go back to the example of a self-driving car. The car presumably has a camera (or multiple cameras). Assume that the car approaches an intersection and sees a traffic sign. Perhaps this traffic sign is a stop sign. How can the adversary perturb the input. Perhaps the adversary can "hack" the car itself. But a much simpler attack is to perturb the actual physical stop sign. Is it possible to change the stop sign slightly so that a human will still recognize it as a stop sign but that the car will likely mistake it for a different sign? Note that this is quite more complicated than our original set-up. We are not given a particular input $\mathbf{x}$ but a whole family of such inputs – this family comes about since the care might approach the same stop sign from various slightly different angles, distances, and under slightly

different ambient lighting conditions. And we would like the same attack (the same physical manipulation of this stop sign) to work under a broad set of those conditions. Even in this cases it has been shown that adversarial changes can be found! Again, this is quite non-trivial! Figure 2 shows what a perturbed stop sign might look like. Note that for a human the change is visible but it is unlikely that it will cause confusion.

Figure 2: To stop or not to stop.

Why Some ML Algorithms Might Not Be Robust – Non-Robust Features

The following example illustrates one reason why a ML algorithm might learn a classification rule that has low standard risk but high adversarial risk.
This is a toy example, but the basic idea is sound — the ML algorithm might rely on a large set of "non-robust" features that can be easily tricked by perturbations.

Consider a binary classification task. We have $y \in \{\pm 1\}$. Let $\hat{\mathbf{x}}$ be the input vector. Assume that after applying a suitable transform we get the new input vector $\mathbf{x}$ that has the following simple structure.

We have $\mathbf{x} = (x_1, \cdots, x_D)$, where $x_i = a_i y + Z_i$, $i = 1, \cdots, D$, where Z_i is Gaussian zero-mean and unit-variance noise which is independent for each component. Further, $a_1 = 1$ and for $i = 2, \cdots, D$, we have $a_i = \sqrt{\frac{\log(D)}{D-1}}$. The exact values for a_i are not so important and are chosen simply for convenience. What is important is that the first component contains a strong signal component, whereas the other features have a very weak such component. Strong versus weak is with respect to the strenght of the noise that is added (zero-mean Gaussian of unit variance). We will say that the first feature is *robust* whereas the other features are not robust.

To summarize, each of the D components is a scaled and noisy version of the label and all components represent conditionally independent observations. Further, the first component contais a strong signal component. The remaining $D-1$ features contain extremely weak signal components but there are many of them (we assume that D is large). Finally, assume that we are given the prior on the label $p(y)$ and that it is uniform.

Assume at first that we are interested in the best classifier without adversarial perturbations, i.e., the classifier with the smallest possible risk (error probability). This is the Bayes classifier, i.e., we should compute the posterior probability

and then choose that label that maximizes the posterior:

$$\begin{aligned}\text{argmax}_{\hat{y}\in\{\pm 1\}} p(y \mid \mathbf{x}) &= \text{argmax}_{\hat{y}\in\{\pm 1\}} \frac{p(\mathbf{x} \mid y)p(y)}{p(\mathbf{x})} \\ &= \text{argmax}_{\hat{y}\in\{\pm 1\}} \prod_{i=1}^{d} p(\mathbf{x}_i \mid y).\end{aligned}$$

In the last step we have used the fact that under our model the observations are conditionally independent so that we get a product of the probabilities and that we have a uniform prior. This can further be simplified to

$$\begin{aligned}\text{argmax}_{y\in\{\pm 1\}} p(y \mid \mathbf{x}) &= \text{argmax}_{\hat{y}\in\{\pm 1\}} \prod_{i=1}^{d} p(\mathbf{x}_i \mid \hat{y}) \\ &= \text{argmax}_{\hat{y}\in\{\pm 1\}} \log \prod_{i=1}^{d} p(\mathbf{x}_i \mid \hat{y}) \\ &= \text{argmax}_{\hat{y}\in\{\pm 1\}} \sum_{i=1}^{d} \log p(\mathbf{x}_i \mid \hat{y}) \\ &= \text{argmax}_{\hat{y}\in\{\pm 1\}} \sum_{i=1}^{d} \log \frac{1}{\sqrt{2\pi}} e^{-\frac{1}{2}(\mathbf{x}_i - \hat{y}a_i)^2} \\ &= \text{argmin}_{\hat{y}\in\{\pm 1\}} \sum_{i=1}^{d} (\mathbf{x}_i - \hat{y}a_i)^2 \\ &= \text{argmin}_{\hat{y}\in\{\pm 1\}} \sum_{i=1}^{d} (\mathbf{x}_i^2 - 2\mathbf{x}_i\hat{y}a_i + \hat{y}^2 a_i^2) \\ &= \text{argmax}_{\hat{y}\in\{\pm 1\}} \hat{y} \sum_{i=1}^{d} \mathbf{x}_i a_i.\end{aligned}$$

Recall that we observe $\mathbf{x}$ which is the vector $y\big(a_1 = 1, a_2 = \sqrt{\frac{\log(D)}{D-1}}, \cdots, a_D = \sqrt{\frac{\log(D)}{D-1}}\big)$ under Gaussian noise with iid zero-mean components and unit variance in each dimension. Therefore the expression that we maximize over $\hat{y}$ above is equal to

$$\hat{y}y(\sum_{i=1}^{D} a_i^2) + \hat{y}\sum_{i=1}^{D} a_i Z_i = \hat{y}y(1 + \log(D)) + \hat{y}Z,$$

where Z is a Gaussian noise with variance $(\sum_{i=1}^{D} a_i^2) = 1 + \log(D)$. Scaling everything by $1/(1 + \log(D))$, we see that this is equivalent to observing the signal $y = \pm 1$ under a zero-mean Gaussian noise with variance $1/(1 + \log(D))$. Hence as D grows the variance tends to zero and our error probability will go to zero as well. In other words, if we train our ML algorithm well then we can hope to get close to zero standard risk when the dimension D grows.

But assume now that we allow the adversary to move the point $\mathbf{x}$ into $\tilde{\mathbf{x}}$ and that our norm is ℓ_∞ with $\varepsilon = 2\sqrt{\frac{\log D}{D-1}}$. In this case the adversary can do the following. She can first make an optimal decision based on the observation. As we have seen, with high probability she will know the correct label. She can then can move each of the non-robust features $i = 2, \cdots, D$ into the wrong direction by an amount $2\sqrt{\frac{\log D}{D-1}}$. This means that she can in effect flip the non-robust features. She can also move the first component somewhat but this has almost no effect. If we ignore the effect on the first component we see that with this change the classifier will misclassify every single sample with high probability!

In summary, we have seen an example where the optimal standard risk is essentially zero but the adversarial risk of the same classifier is almost 1. This is as bad as it gets. And we have seen that this is due to the fact that the classifier relied heavily on many very weak features that were easy to perturb.

Could we have constructed a more robust classifier? Yes, certainly, but at a price. Assume that we build a classifier based on the first feature only. The best classifier in this case is again a Bayes classifier. A little bit of thought shows that it is given by simply taking the sign of $\mathbf{x}_1$. What is the risk of this classifier? We have a label that is either $+1$ or -1. We add a zero-mean unit-variance Gaussian random variable to it and ask what is the probability that this noise changes the sign. A little bit of thought shows that the incurred error probability in this case is equal to

$$\frac{1}{\sqrt{2\pi}} \int_1^\infty e^{-\frac{1}{2}x^2} dx \sim 0.16.$$

In words, this classifier incurs a standard risk of about sixteen percent. This is much higher than the standard risk of essentially zero we saw above. But in this case the risk almost does not change if we consider the adversarial setting since in our threat model we allow the adversary to move each component by only a very small amount. So we see in this model a trade-off between a small standard risk and a small adversarial risk. We cannot get both.

The above example might look a little construed and also it might not be clear how much of this is due to the fact that we allow the adversary to change the components in an ℓ_∞

sense. Indeed, it is more challenging to find simple examples if the threat model consist of changes in ℓ_2. But similar ideas still apply.
The idea for the above model and further details come from the paper "Robustness May Be at Odds with Accuracy," by Tsipras, Dimitris, Santurkar, Shibani, Engstrom, Logan, Turner, Alexander, and Madry, Aleksander.

Why Some ML Algorithms Might Not Be Robust – The Curse of Dimensionality – Again!

THIS SECTION STILL NEEDS TO BE COMPLETED.
Let us talk about a simple setting to see why in high dimension adversarial examples cannot be completely avoided.
We consider a binary classification example. We are operating in high dimensions, lets say $\mathcal{R}^{500}$, i.e., $D = 500$. Our data is perfectly separable. For $y = -1$ the data is distributed uniformly on the surface of a sphere of radius 1 and for $y = 1$ the dat is unformly distributed on the surface of a sphere of radius 1.3.
Empirically, if you get a large number of samples (lets say one million) and train a NN on this data then you likely get a network that correctly classifies each and every single element of the training set and further has a fairly small standard risk. But if you allow some adversarial perturbations the adversarial risk is very high. How can this happen? A perturbation of .. seems small.
This example is sufficiently simple that we can analyze it

precisely. The basic idea is the following. Even if we get a large number of samples, due to the large dimension, there are directions (think of a small cone along this direction) along which we have not received any data. Assume that we are even told by a genie that we should look for a decision boundary of the form

$$\sum_{i=1}^{D} \alpha_i x_i^2 = \tau,$$

i.e., we know that the best decision boundary is an ellipsoide but we are not told that it is sphere. Since there are some directions along which we do not see any data, there is nothing in our learning that could help us decide between a decision boundary whose axis in this direction might extend beyond the spherical one and the spherical one itself. All of them will perfectly separate the given data and also all of them will likely have a small standard risk.
But – and here is where the curse of dimensionality hits – this small risk is blown up to a very large adversarial risk due to the large D. How can this happen?

Constructing a Robust Classifier from Scratch – Adversarial Training

So far we have assumed that we are given a classifier and we discussed what might happen if we allow adversarial changes to the input. But we can take a more pro-active approach.
First, we can include the aim for adversarial robustness in the training phase. This is what we discuss now. Second,

given a classifier we can ask if we can modify it to make it more robust. We will briefly discuss this second approach in the next section. In practice we might want to apply both techniques. We will be *very* brief.

The approach taken in the paper "Towards Deep Learning Models Resistant to Adversarial Attacks" by Madry, Aleksander, Makelov, Aleksandar, Schmidt, Ludwig, Tsipras, Dimitris, and Vladu is the following perhaps most natural idea. Rather than minimizing the usual cost function

$$\min_{\Theta} \ \big[\mathcal{L}(f) = \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\big[\mathbb{1}_{\{f_\Theta(\mathbf{x})\neq y\}}\big]\big]$$

over all parameters Θ of the model, why not directly minimize what matters, namely

$$\min_{\Theta} \ \big[\mathcal{R}(f,\varepsilon) = \min_{\Theta} \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\big[\max_{\tilde{\mathbf{x}}:\|\mathbf{x}-\tilde{\mathbf{x}}\|\leq\varepsilon} \mathbb{1}_{\{f(\tilde{\mathbf{x}})\neq y\}}\big]\big].$$

This leads to a min-max formulation. As written, this function is not easy to optimize for several reasons. The first is that it is not smooth (due to the indicator function). Therefore let us instead look at the problem

$$\min_{\Theta} \ \mathbb{E}_{(\mathbf{x},y)\sim\mathcal{D}}\big[\max_{\tilde{\mathbf{x}}:\|\mathbf{x}-\tilde{\mathbf{x}}\|\leq\varepsilon} (\tfrac{1}{2} - y(g(\tilde{\mathbf{x}}) - \tfrac{1}{2}))\big].$$

Here we assumed that we deal with a binary classification problem, i.e., $y \in \{\pm 1\}$ and that the classifier $f(\mathbf{x})$ has the form (1) so that $(\frac{1}{2} - y(g(\tilde{\mathbf{x}}) - \frac{1}{2}))$ is the predicted probability of the incorrect label. Now we are dealing with a smooth function.

We do not have access to the distribution but instead we

have a sample $\mathcal{S}$. Therefore the equivalent problem is

$$\min_{\Theta} \sum_{n=1}^{N} \max_{\tilde{\mathbf{x}_n}:\|\mathbf{x}_n-\tilde{\mathbf{x}_n}\|\leq\varepsilon} \left(\tfrac{1}{2} - y_n(g(\tilde{\mathbf{x}_n}) - \tfrac{1}{2})\right).$$

It is not completely obvious how to minimize this. It turns out that the following is correct. Compute for each $\mathbf{x}_n$ the worst perturbation $\tilde{\mathbf{x}}_n$.Then take the gradient of this expression and move towards the negative gradient direction. Of course, this is computationally intensive since for each sample and each iteration we need to find the worst case perturbation. This training algorithm is called *adversarial training*.

Making an Existing Classifier Robust – Randomized Smoothing

In some instances we might be handed a classifier that has small standard risk but might be prone to adversarial errors and are asked to make it more robust rather than learning a new classifier from scratch. To date the perhaps most promising idea of accomplishing this is *randomized smoothing*.

Let f be the given classifier. Assume that it maps $\mathbb{R}^D$ to $\mathcal{Y}$, the set of labels. From this we derive the *smoothed* classifier, call it g. This new smoothed classifier g maps an input $\mathbf{x}$ to that class c that is most likely returned by f if presented the input $\mathbf{x} + \mathbf{z}$, where $\mathbf{z}$ is a vector of iid zero-mean random Gaussian variables with a variance of σ^2 per component. This idea was introduced by Lecuyer, M., Atlidakis, V., Geambasu, R., Hsu, D.,and Jana, S. in the paper

entitled "Certified robustness to adversarial examples with differential privacy". It was shown to work well for various test data sets like ImageNet. There are several variants of this. E.g., rather than using this probabilistic definition using Gaussians we could average over a ball of radius $\sqrt{D}\sigma$ (with a uniform distribution). The effect would be more or less the same. This resulting training algorithm variants are called *robust training*.

The basic idea why this adds robustness is the following. Consider e.g. the binary case. Take a point $\mathbf{x}$ and assume that for the original classifier f lets say the label $y = 1$ is much more likely to be returned by f if we give it the point $\mathbf{x} + \mathbf{z}$. Roughly speaking, this means that if we consider a ball of radius $\sqrt{D}\sigma$ around the point $\mathbf{x}$ then "most points" within this ball are classified by f to be 1. It is then intuitive, and can be shown rigorously, that if we move the point $\mathbf{x}$ a little bit and consider again a ball of radius $\sqrt{D}\sigma$ around this new point that still most points are classified by f as 1. This is true since the two balls will have a large overlap and so should have similar averages. Therefore, g will likely not change its classification between two points that are close and this is exactly what robustnes asks for.

Not all is perfect. The averaging (smoothing) can increase the standard risk considerably. I.e., there might be a considerable price to pay for the added robustness.