Probabilistic Graphical Models
Are Generative Classifiers More Robust to Adversarial Attacks?

Manal Akhannouss, ENS Paris-Saclay
manal.akhannouss@eleves.enpc.fr

Paul Barbier, ENS Paris-Saclay
paul.barbier@eleves.enpc.fr

Alexandre Lutt, ENS Paris-Saclay
alexandre.lutt@eleves.enpc.fr

I. Introduction

II. Models

II.1. Discriminative versus generative models

Let us consider a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$ of N samples, where $\boldsymbol{x}_i \in \mathbb{R}^d$ is a d-dimensional feature vector and $\boldsymbol{y}_i \in \mathcal{Y}$ is the corresponding label. A discriminative classification model (or discriminative classifier) aims to estimate the conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$, *i.e.* the probability that the label $\boldsymbol{y}$ is associated to the feature vector $\boldsymbol{x}$. On the other hand, a generative classification model (or generative classifier) aims to estimate the joint probability $p(\boldsymbol{x}, \boldsymbol{y})$, *i.e.* the probability of observing the feature vector $\boldsymbol{x}$ and the label $\boldsymbol{y}$ at the same time. Both models can be used for classification purposes, *i.e.* can be used to predict the label $\boldsymbol{y}$ associated to a feature vector $\boldsymbol{x}$, but with very different interpretations. The discriminative classifier will directly estimate the conditional probability $p(\boldsymbol{y}|\boldsymbol{x})$, while the generative classifier will estimate the joint probability $p(\boldsymbol{x}, \boldsymbol{y})$ for each possible value of $\boldsymbol{y}$, and then use the Bayes rule to estimate the conditional probability $p(\boldsymbol{y}|\boldsymbol{x}) = \dfrac{p(\boldsymbol{x}|\boldsymbol{y})p(\boldsymbol{y})}{p(\boldsymbol{x})}$.

One of the most common generative classifier is Naive Bayes. This simple model assumes a factorised distribution $p(\boldsymbol{x}|\boldsymbol{y}) = \displaystyle\prod_{i=1}^{d} p(\boldsymbol{x}_i|\boldsymbol{y})$, which means that the features are independent given the label. This assumption is often far from being verified in practice for image datasets. For this reason, in the following, we will follow the path of [2] and use a latent-variable model $p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ to design our generative classifier. Note that this model does not assume a factorised distribution for $p(\boldsymbol{x}|\boldsymbol{y})$, since in this case $p(\boldsymbol{x}|\boldsymbol{y}) = \dfrac{\int p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})d\boldsymbol{z}}{\int p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})d\boldsymbol{x}d\boldsymbol{y}}$. In order to fully define a latent-variable model, we need to explicitely chose a structure for $p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$. At this point, several choices are possible:

$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p_{\mathcal{D}}(\boldsymbol{x})p(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z}) \tag{DFX}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z}) \tag{DFZ}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p_{\mathcal{D}}(\boldsymbol{x})p(\boldsymbol{z}|\boldsymbol{x})p(\boldsymbol{y}|\boldsymbol{z}) \tag{DBX}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{z})p(\boldsymbol{y}|\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{z}) \tag{GFZ}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p_{\mathcal{D}}(\boldsymbol{y})p(\boldsymbol{z}|\boldsymbol{y})p(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{z}) \tag{GFY}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{z})p(\boldsymbol{y}|\boldsymbol{z})p(\boldsymbol{x}|\boldsymbol{z}) \tag{GBZ}$$
$$p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}) = p_{\mathcal{D}}(\boldsymbol{y})p(\boldsymbol{z}|\boldsymbol{y})p(\boldsymbol{x}|\boldsymbol{z}) \tag{GBY}$$

In those acronyms, D stands for *discriminative*, G stands for *generative*, F stands for *fully-connected graph* (see 1), and the last letter indicates on which variable we assume a prior distribution (determined with $\mathcal{D}$ in the case of X and Y). In our case, we will focus on the DFZ and GFZ structures, in order to be able to compare the discriminative and generative approaches, but everything that we will see can easily be extended to the other structures.

Figure 1. Examples of latent-variable models architecture

II.2. Classifiers architecture

As mentionned above, we will consider two different classifiers. The first one will be a simple discriminative classifier (with DFZ structure), while the second one will be a generative classifier (with GFZ structure).

III. Adversarial attacks

In this section, we will present the different methods we used to create adversarial attacks on our models. These methods can be divided into two categories: white box attacks and black box attacks, depending on the knowledge of the attacker regarding the model. If the attacker has access to the model's parameters and architecture, we will talk about white box attacks. Otherwise, we will talk about black box attacks.
In all cases, we aim to design attacks which are as imperceptible as possible to the human eye, while changing the predicted label of the model. In other words, we aim to solve the following optimization problem:

$$\begin{aligned} \min_{\boldsymbol{\eta}} \quad & \|\boldsymbol{\eta}(\boldsymbol{x})\|_2 \\ \text{s.t.} \quad & \underset{\boldsymbol{y}\in\mathcal{Y}}{\text{argmax}}\ (p(\boldsymbol{y}|\boldsymbol{x}+\boldsymbol{\eta}(\boldsymbol{x}))) \neq \underset{\boldsymbol{y}\in\mathcal{Y}}{\text{argmax}}\ p(\boldsymbol{y}|\boldsymbol{x}) \end{aligned}$$

where $\boldsymbol{\eta}$ is the adversarial perturbation, $\boldsymbol{x}$ is the original sample, and $p(\boldsymbol{y}|\boldsymbol{x})$ is the conditional probability of the label $\boldsymbol{y}$ given the sample $\boldsymbol{x}$. Note that as we are considering a multiclass classification problem, we will implement the different attacks as one-vs-all attacks, *i.e.* we will create adversarial examples to change the prediction of the model for each class independently.

In order to compare the different algorithmic solutions to this problem, we will keep using the Fashion-MNIST dataset mentionned above, and we will compare the different approches by computing the accuracy of the model on the adversarial examples, as well as the average robustness $\rho = \frac{1}{n_{\text{samples}}}\sum_{i=1}^{n_{\text{samples}}}\frac{\|\boldsymbol{\eta}(\boldsymbol{x}_i)\|_2}{\|\boldsymbol{x}_i\|_2}$ of the model to the adversarial perturbations.

III.1. White box attacks

III.1.1 Fast Gradient Sign (FGS) method

We first implemented the Fast Gradient Sign (FGS) method [1], which is a simple and efficient method to create adversarial attacks. Let us consider a model with a model loss function J_θ and a sample $(\boldsymbol{x}, \boldsymbol{y})$. The FGS method consists in adding a perturbation $\boldsymbol{\eta}$ to the input $\boldsymbol{x}$, with $\boldsymbol{\eta} = \varepsilon \times \text{sign}(\nabla_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, \boldsymbol{y}))$, where ε is a hyperparameter which controls the magnitude of the perturbation. For more details, our implementation of the Fast Gradient Sign method is described in 1. This method is very simple to implement, and cheap to run (since it only requires one gradient computation), but it comes with a practical drawback; the perturbations are often very visible to the human eye, which makes them less realistic. This comes from the fact that the ε hyperparameter is often chosen to be too large, in order to ensure that the perturbation is large enough to fool the model.

III.1.2 DeepFool method

In order to create more subtle perturbations, we then implemented the DeepFool algorithm. This algorithm was first introduced in [3], as a way to create small adversarial examples for deep neural networks. The idea is to iteratively compute the minimal perturbation $\boldsymbol{\eta}$ which is required to change the prediction of the model. More precisely, let us consider a model with a model loss function J_θ and a sample $(\boldsymbol{x}, \boldsymbol{y})$. The DeepFool algorithm consists in iteratively computing the minimal perturbation $\boldsymbol{\eta}$ which is required to change the prediction of the model, with $\boldsymbol{\eta} = \frac{J_\theta(\boldsymbol{x}, \boldsymbol{y})}{\|\nabla_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, \boldsymbol{y})\|_2^2}\nabla_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, \boldsymbol{y})$. The idea comes from the fact that, if the model is linear, the decision boundary is a hyperplane of equation $\boldsymbol{w}^T\boldsymbol{x} + b = 0$, and the minimal perturbation required to change the prediction of the model is the orthogonal projection of the sample on the decision boundary, *i.e.* $\boldsymbol{\eta} = -\frac{\text{sign}(\boldsymbol{w}^T\boldsymbol{x}+b)}{||\boldsymbol{w}||_2^2}\boldsymbol{w}$. In the non-linear case, the decision boundary is not a hyperplane anymore, but this heuristic still works well in practice. More precisely, our implementation of the DeepFool algorithm is described in 2 (where we can chose another p-norm to optimise on, even if in practice we mostly focused on the case $p = 2$). Obviously, this algorithm is more expensive to run than the FGS method (since it requires several gradient computations), but it is also more efficient since it creates smaller perturbations.

III.2. Black box attacks

IV. Experimental setup

IV.1. Attacks detection

IV.2. Models training

IV.3. Attacks benchmark

IV.4. Attacks detection

V. Results

V.1. Accuracy

V.2. Robustness to perturbations

V.3. Attacks detection

VI. Conclusion

VII. Appendix

VII.1. Fast Gradient Sign algorithm

VII.2. DeepFool algorithm

Algorithm 1: Fast Gradient Sign method

```
Input: $f_\theta$, $\boldsymbol{x}$, $\boldsymbol{y}$, $\varepsilon$
Output: $\boldsymbol{\eta}$
1  for $k \neq y$ do
2      $\boldsymbol{g}_k \leftarrow \boldsymbol{\nabla}_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, k)$
3      $\boldsymbol{\eta}_k \leftarrow \varepsilon \times \text{sign}(\boldsymbol{g}_k)$
4  end
5  $\boldsymbol{\eta} \leftarrow \underset{k, f_\theta(\boldsymbol{x}+\boldsymbol{\eta}_k)\neq y}{\text{argmin}} \|\boldsymbol{\eta}_k\|_2$
```

Algorithm 2: DeepFool algorithm

```
Input: $f_\theta$, $\boldsymbol{x}$, $\boldsymbol{y}$, $\varepsilon$, $p > 1$
Output: $\boldsymbol{\eta}$
1  $q \leftarrow \frac{p}{p-1}$
2  $i \leftarrow 0$
3  $\boldsymbol{\eta} \leftarrow 0$
4  $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$
5  while $f_\theta(\boldsymbol{x}_i) = y$ do
6      $i \leftarrow i + 1$
7      for $k \neq y$ do
8          $\boldsymbol{w}_k \leftarrow \boldsymbol{\nabla}_{\boldsymbol{x}} J_\theta(\boldsymbol{x}_i, k) - \boldsymbol{\nabla}_{\boldsymbol{x}} J_\theta(\boldsymbol{x}_i, y)$
9          $f_k \leftarrow J_\theta(\boldsymbol{x}_i, k) - J_\theta(\boldsymbol{x}_i, y)$
10     end
11     $l \leftarrow \underset{k\neq y}{\text{argmin}} \frac{|f_k|}{||\boldsymbol{w}_k||_q}$
12     $\boldsymbol{\eta}_i \leftarrow \frac{|f_l|}{||\boldsymbol{w}_l||_q^q} |\boldsymbol{w}_l|^{q-1} \times \text{sign}(\boldsymbol{w}_l)$
13     $\boldsymbol{x}_{i+1} \leftarrow \boldsymbol{x}_i + \boldsymbol{\eta}_i$
14     $\boldsymbol{\eta} \leftarrow \boldsymbol{\eta} + \boldsymbol{\eta}_i$
15 end
```

References

[1] J. Goodfellow, I., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. 2

[2] Li, Y., Bradshaw, J., and Sharma, Y. Are generative classifiers more robust to adversarial attacks? 1

[3] Moosavi-Dezfooli, S.-M., Fawzi, A., and Frossard, P. Deepfool: a simple and accurate method to fool deep neural networks. 2