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

In this section, we will present the different models we used for our experiments. More precisely, we will introduce generative models, present the two classifiers we used for our experiments as well as their respective architecture. Then, we will present the dataset that we worked with, and provide details about the training procedure of our 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 $p(\boldsymbol{x}|\boldsymbol{y}) = \prod_{i=1}^{d} p(\boldsymbol{x}_i|\boldsymbol{y})$, 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 [3] 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*, 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.

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). More specifically, we will consider the following architectures (all of them include dropout during training for regularization purposes):

- **Discriminative classifier:** In this case, we use two neural networks to approximate $p(\boldsymbol{x}|\boldsymbol{z})$ and $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$ respectively. More precisely, we use a convolutionnal neural network with 3 convolutions layers with ReLU activations, followed by 3 dense layers, to approximate $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$. On the other hand, $p(\boldsymbol{x}|\boldsymbol{z})$ is obtained after a sequence (with dropout during training) of 2 dense layers and 3 transposed convolutions. Note that we use a transposed convolution instead of a simple convolution, in order to be able to reconstruct the original image size.

- **Generative classifier:** This case differs from the previous one in the sense that we use a simple 3-layers MLP to approximate $p(\boldsymbol{y}|\boldsymbol{z})$ as well as a sequential model with 2 dense layers and 3 transposed convolutions for $p(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{z})$.

Note that in both cases, we use the amortised approximate posterior $q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{z})$ for training. More specifically, we learn both $p(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z})$ and $q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{z})$ by maximizing the variational lower bound as in [2]:

$$\mathbb{E}_{\mathcal{D}}[\mathcal{L}_{VI}(\boldsymbol{x}, \boldsymbol{y})] = \frac{1}{N}\sum_{i=1}^{N}\mathbb{E}_q\Big[\log\frac{p(\boldsymbol{x}_n, \boldsymbol{y}_n, \boldsymbol{z}_n)}{q(\boldsymbol{z}_n|\boldsymbol{x}_n, \boldsymbol{y}_n)}\Big]$$

We also use a convolutionnal neural network similar to the one used to model $p(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{z})$ to approximate $q(\boldsymbol{z}|\boldsymbol{x}, \boldsymbol{y})$.

II.3. Dataset

For all of our experiments, we used the Fashion MNIST dataset, which is a dataset of 70 000 grayscale images of size 28×28 pixels, each associated to one of 10 classes. The dataset is divided into a training set of 60 000 images and a test set of 10 000 images. The reason for this choice comes from the very high accuracies obtained by [3] on the original MNIST dataset. In order to make the classification task more challenging, we decided to use the Fashion MNIST dataset instead, which is very similar to the original MNIST dataset, but with more complex images (fashion items *versus* handwritten digits).

II.4. Training

In order to train our models, we used the Adam optimizer with a learning rate of 10^{-3}, a batch size of 100, and we trained our models for 20 epochs on a T400 GPU. Note that we did not use any data augmentation technique, since we wanted to focus on the robustness of our models to adversarial attacks, and not on the robustness of the data augmentation techniques themselves.

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:

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

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 $\rho = \frac{1}{n_{\text{samples}}} \sum_{i=1}^{n_{\text{samples}}} \frac{\|\boldsymbol{\eta}_i\|_2}{\|\boldsymbol{x}_i\|_2}$ 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 $\boldsymbol{\eta} = \varepsilon \times \text{sign}(\boldsymbol{\nabla}_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, \boldsymbol{y}))$, 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 "often enough".

III.1.2 DeepFool method

In order to create more subtle perturbations, we then implemented the DeepFool algorithm. This algorithm was first introduced in [4], 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 $\boldsymbol{\eta} = \frac{J_\theta(\boldsymbol{x}, \boldsymbol{y})}{\|\boldsymbol{\nabla}_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, \boldsymbol{y})\|_2^2} \boldsymbol{\nabla}_{\boldsymbol{x}} J_\theta(\boldsymbol{x}, \boldsymbol{y})$.

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

III.3. Attacks detection

In order to detect adversarial examples, we decided to make use of the logits of the model. More precisely, we will use the fact that the logits of the models are often quite different for adversarial examples. In order to do so, we will use the logits-based detection method introduced in [3]. The main idea of this method is that the logits values $(\log p(\boldsymbol{x}, \boldsymbol{y}_c))_{c \in C}$ of a generative classifier correspond to the log probability of generating the sample $\boldsymbol{x}$ for each label $\boldsymbol{y}_c$.

The main idea behind logit-based attacks detection is to reject inputs using the joint density $p(\boldsymbol{x}, \boldsymbol{y})$ of the generative classifier, *i.e.* to reject a sample x if $p(\boldsymbol{x}, F(\boldsymbol{x}))$ (where $y = F(\boldsymbol{x})$ is the label predicted by the model for the sample $\boldsymbol{x}$) is too low. As usual for probability tresholds, we use the logarithmic scale, and we reject a sample $\boldsymbol{x}$ if $\log p(\boldsymbol{x}, F(\boldsymbol{x})) < \delta_{\boldsymbol{y}}$. We compute the tresholds $(\delta_{\boldsymbol{y}_c})_{c \in C}$ using the well classified samples of the training set with the following formula: $\delta_{\boldsymbol{y}_c} = \mu_c + \alpha \sigma_c$, where μ_c and σ_c are the mean and standard deviation of the log-probability of the well-classified samples of class $\boldsymbol{y}_c$ in the training set, and α is a hyperparameter which controls the tresholds scale.

IV. Experimental setup

In this section, we will discuss the different experiments we ran in order to compare the robustness of the discriminative and generative classifiers to adversarial attacks. More precisely, we will first present the performances of our models, then we will compare the different attacks we used to create adversarial examples, and finally we will the results of a detection method specially designed to detect adversarial examples with generative classifiers.

IV.1. Models performances

IV.2. Attacks benchmark

IV.3. 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

Algorithm 1: Fast Gradient Sign method

```
Input: $f_\theta$, $\boldsymbol{x}$, $\boldsymbol{y}$, $\varepsilon$
Output: $\boldsymbol{\eta}$
1 $\boldsymbol{\eta} \leftarrow \varepsilon \times \text{sign}(\nabla_{\boldsymbol{x}} J_\theta(\boldsymbol{x}))$
```

Algorithm 2: DeepFool algorithm

```
Input: $f_\theta$, $\boldsymbol{x}$, $\boldsymbol{y}$, $p > 1$
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          $\boldsymbol{w}_k \leftarrow \nabla_{\boldsymbol{x}} J_\theta(\boldsymbol{x}_i, k) - \nabla_{\boldsymbol{x}} J_\theta(\boldsymbol{x}_i, y)$
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
```

VII.2. DeepFool algorithm

References

[1] J. Goodfellow, I., Shlens, J., and Szegedy, C. Explaining and harnessing adversarial examples. 3

[2] Kingma, D. P., and Welling, M. Auto-encoding variational bayes. 2

[3] Li, Y., Bradshaw, J., and Sharma, Y. Are generative classifiers more robust to adversarial attacks? 1, 2, 3

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