"Are Generative Classifiers More Robust to Adversarial Attacks?" paper (2018)

Review of their Deep Bayes model

Context

- Deep neural network can easily be fooled by some imperceptible perturbation, even black-box networks.
- In 2018: only research on discriminative classifiers, not on generative ones.
- Popular generative classifiers such as naive Bayes are not well suited for real world data.

The paper solution

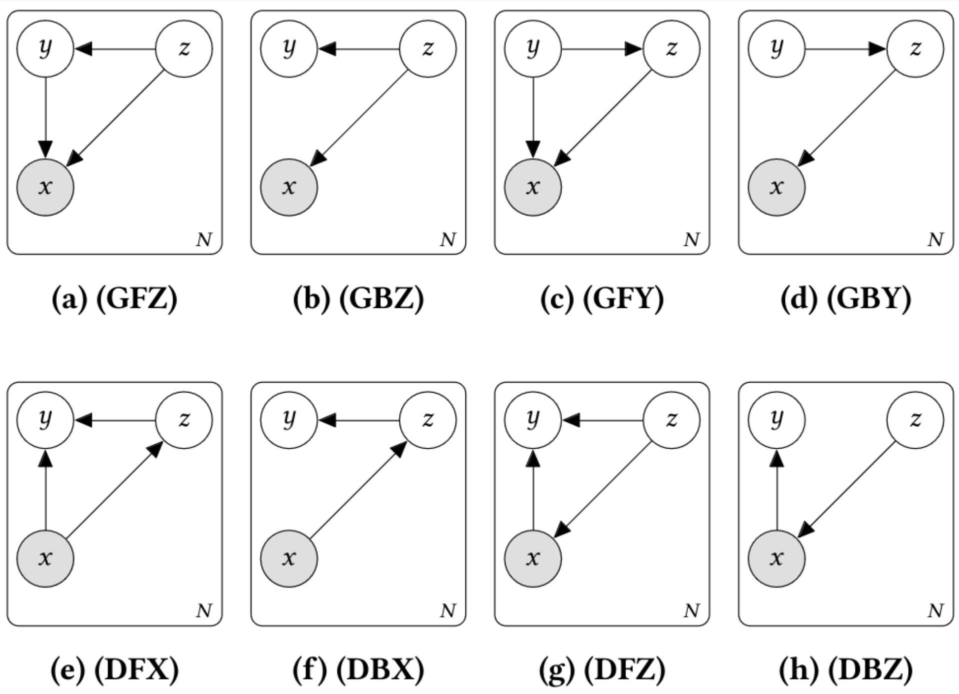
- They improve Naive Bayes with a latent variable. They create Deep Bayes, a Deep Latent Variable Model trained as a VAE.
- They investigate 3 adversarial attack detection methods, and compare the results depending on the Deep Bayes architecture.

My contributions

- Further graphical models explanations.
- More details on Deep Bayes construction.
- Pseudo-code for Deep Bayes.

Latent variable z

- In my review, I explain why there are 8 interesting ways of adding a latent variable z:



(a) (GFZ) (b) (GBZ) (c) (GFY) (d) (GBY)

(e) (DFX) (f) (DBX) (g) (DFZ) (h) (DBZ)

- We then have 8 different factorizations for $p(x, z, y)$, which leads to 8 different Deep Bayes models.
- The paper shows that the architecture has an impact on the robustness to adversarial attacks.

Deep Bayes

After some computations detailed in my review we end up with a formula for the class $y*$ of a given $x*$:

$$y* \sim p(y|x^*) \approx \mathrm{softmax}_{c=1}^{C}\left(\log \frac{1}{K}\sum_{k=1}^{K}\frac{p(x^*, z_c^k, y_c)}{q(z_c^k|x^*, y_c)}\right)$$

where $q(z|x, y)$ approximates $p(z|x, y)$ and $z_c^k \sim q(z|x^*, y_c)$.

Training

As for a VAE, we maximize the variational lower-bound:

$$\mathbb{E}_{\mathcal{D}}\left[\mathcal{L}_{\mathrm{VI}}(x, y)\right] = \frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_q\left[\log\frac{p(x_n, z_n, y_n)}{q(z_n|x_n, y_n)}\right]$$

where $\mathcal{D}$ is the dataset.

Example: GFZ algorithm

For (GFZ), we have $p(x, z, y) = p(z)p(y|z)p(x|z, y)$:

Algorithm GFZ Deep Bayes

Require: $x^* \in \mathbb{R}^D$
Ensure: $y* \sim p(y|x^*)$
logpxy ← []
for $c = 1, \dots, C$ **do**
# computation of $q(z_c^k|x^*, y_c)$:
$z \leftarrow$ **VAE_encoder**(x^*, y_c) ▷ $z_c^k \sim q(z|x^*, y_c)$
log_q ← LogGaussProb(z, μ, σ) ▷ $q(z_c^k|x^*, y_c)$

computation of $p(x^*, z_c^k, y_c)$:
prior $p(z)$
log_pz ← LogGaussProb(z) ▷ $p(z_c^k)$
intermediate $p(y|z)$
y_logit ← **MLP_pyz**(z) ▷ $p(y|z_c^k)$
log_pyz ← softmax_logits$(y_c$, y_logit$)$ ▷ $p(y_c|z_c^k)$
Likelihood $p(x|z, y)$
x ← **MLP_pxzy**(z, y) ▷ $x \sim p(x|z_c^k, y_c)$
mux, sigx ← stat(x)
log_pxzy ← LogGaussProb$(x^*$, mux, sigx$)$ ▷ $p(x^*|z_c^k, y_c)$

proba ← log_pxzy + log_pyz + log_pz − log_q
# proba = $\log p(x^*, y_c) = \log\left(\frac{p(x^*|z_c^k, y_c)p(y_c|z_c^k)p(z_c^k)}{q(z_c^k|x^*, y_c)}\right)$
logpxy.append(proba)
end for
Return softmax$_{c=1}^{C}$(logpxy)

Mathis Embit

mathis.embit@ens-paris-saclay.fr

École normale supérieure Paris-Saclay

MATHÉMATIQUES VISION APPRENTISSAGE