Constrained Adversarial Networks

Andrea Passerini

August 29, 2018

We would like to incorporate constraints to GANs in an implicit way, by an additional teacher penalizing outputs which do not satisfy the constraints.

[TODO] Why do we want to incorporate the constraints?

The idea is to combine two recent works: Boundary-seeking GANs [2] for dealing with discrete outputs, and posterior regularization [1] for implicitly enforcing logic constraints to deep nets [3].

1 Boundary-seeking GANs

The idea is to train the generator to match a target distribution, which at the limit of the perfect discriminator matches the data distribution.

Given the (unknown) data distribution $p_{data}(\mathbf{x})$, the generator distribution $p_g(\mathbf{x})$ and the discriminator $D(\mathbf{x})$, we have that for the perfect discriminator it holds (regardless of how $p_g(\mathbf{x})$ matches $p_{data}(\mathbf{x})$):

$$D^*(\mathbf{x}) = \frac{p_{data}(\mathbf{x})}{p_{data}(\mathbf{x}) + p_g(\mathbf{x})}$$

from which we can derive a formula for the data distribution:

$$p_{data}(\mathbf{x}) = \frac{D^*(\mathbf{x})}{1 - D^*(\mathbf{x})} p_g(\mathbf{x})$$

Given an imperfect discriminator $D(\mathbf{x})$, we can approximate the data distribution as:

$$\tilde{p}_{data}(\mathbf{x}) = \frac{1}{Z} \frac{D(\mathbf{x})}{1 - D(\mathbf{x})} p_g(\mathbf{x})$$

with

$$Z = \sum_x \frac{D(\mathbf{x})}{1 - D(\mathbf{x})} p_g(\mathbf{x})$$

When considering the noise $\mathbf{z}$ in input to the generator, it is convenient to focus on joint distributions (*data* dropped for simplicity):

$$p_g(\mathbf{x}, \mathbf{z}) = g(\mathbf{x}|\mathbf{z}) p(\mathbf{z})$$

and

$$\tilde{p}(\mathbf{x}, \mathbf{z}) = \frac{1}{Z_{|\mathbf{z}}} \frac{D(\mathbf{x})}{1 - D(\mathbf{x})} g(\mathbf{x}|\mathbf{z}) p(\mathbf{z})$$

where $g(\mathbf{x}|\mathbf{z})$ is the generator's output (before discretization) for input $\mathbf{z}$ and $Z_{|\mathbf{z}} = \sum_{\mathbf{x}} \frac{D(\mathbf{x})}{1-D(\mathbf{x})} g(\mathbf{x}|\mathbf{z})$. Learning the generator now amounts at minimizing its Kullback-Leibler divergence to the estimated data distribution:

$$\min_g D_{KL}(\tilde{p}(\mathbf{x}, \mathbf{z}) || p_g(\mathbf{x}, \mathbf{z}))$$

See Appendix A for a detailed derivation of the gradient.

2 Harnessing Deep Neural Networks with Logic Rules

The idea is to jointly train two models: the student ($p(\mathbf{y}|\mathbf{x})$) minimizes a combination of a loss function on supervised data and a loss function on predictions made by the teacher; the teacher ($q(\mathbf{y}|\mathbf{x})$) minimizes the KL divergence to the student model and the number of rule violations.
The student minimization problem is:

$$\min_\theta \sum_{(\mathbf{x},\mathbf{y}) \in \mathcal{D}} \gamma \ell(\mathbf{y}, p_\theta(\mathbf{y}|\mathbf{x})) + (1-\gamma) \ell(q(\mathbf{y}|\mathbf{x}), p_\theta(\mathbf{y}|\mathbf{x}))$$

The teacher minimization problem is:

$$\begin{aligned} \min_q \quad & D_{KL}(q(\mathbf{y}|\mathbf{x}) || p_\theta(\mathbf{y}|\mathbf{x})) + C \sum_{l,g_l} \xi_{l,g_l} \\ s.t. \quad & \lambda_l (1 - E_q[r_{l,g_l}(\mathbf{y}, \mathbf{x})]) \leq \xi_{l,g_l} \quad \forall l, \forall g_l \end{aligned}$$

Here l ranges over the set of rules, g_l over the set of rule groundings on $(\mathbf{x}, \mathbf{y})$, $r_{l,g_l}(\mathbf{y}, \mathbf{x})$ computes the soft rule using e.g. the Lukasiewicz t-norm, and λ_l is the strength of rule l. The problem has a closed-form solution given by:

$$q(\mathbf{y}|\mathbf{x}) \propto p_\theta(\mathbf{y}|\mathbf{x}) \exp\left[- \sum_{l,g_l} C \lambda_l (1 - r_{l,g_l}(\mathbf{y}, \mathbf{x})) \right]$$

The complexity of computing the normalization factor depends on the rules. In the worst case, one needs to resort to approximation strategies (e.g. Gibbs sampling). As a very first approximation, one could employ an importance sampling strategy similar to the one used in BGANs, i.e. sample according to $p_\theta(\mathbf{y}|\mathbf{x})$ and reweight by $\exp\left[- \sum_{l,g_l} C \lambda_l (1 - r_{l,g_l}(\mathbf{y}, \mathbf{x})) \right]$.

3 Possible directions

The first two directions consist in modifying the BGAN framework by providing either the discriminator or the generator with a signal on the satisfaction of the constraints $\Psi(\mathbf{x})$. An alternative direction is to introduce a third network (the teacher T) that is trained to "improve" the objects generated by G by modifying them in order to satisfy the constraints. The generator is trained not only to fool the discriminator, but also to approximate the teacher's distribution (this closely resembles the approach presented in [3]). **[TODO]** Add the reinforcement learning-based direction.

3.1 Constraint-regularized discriminator

$$
\begin{aligned}
l_G &= E[D(G(\mathbf{z}))]_{\mathbf{z}\sim P_\mathbf{z}} \\
l_D &= E[D(\mathbf{x}, \Psi(\mathbf{x}))]_{\mathbf{x}\sim P_\mathbf{x}} + E[1 - D(G(\mathbf{z}), \Psi(G(\mathbf{z})))]_{\mathbf{z}\sim P_\mathbf{z}}
\end{aligned}
$$

The discriminator is provided with both an object $\mathbf{x}$ and a signal on the satisfaction of the constraints $\Psi(\mathbf{x})$. By making the discriminator aware of Ψ, the generator should be pushed to generate objects which satisfy the constraints (as long as those constraints are also satisfied in the training data).

Since l_G is constraint-agnostic, this approach can be used even with constraints that are not differentiable and with constraints that cannot be written (or are difficult to write) in closed form.

3.2 Constraint-regularized generator

$$
\begin{aligned}
l_G &= E[D(G(\mathbf{z}))]_{\mathbf{z}\sim P_\mathbf{z}} + \lambda E[\Psi(G(\mathbf{z}))]_{\mathbf{z}\sim P_\mathbf{z}} \\
l_D &= E[D(\mathbf{x})]_{\mathbf{x}\sim P_\mathbf{x}} + E[1 - D(G(\mathbf{z}))]_{\mathbf{z}\sim P_\mathbf{z}}
\end{aligned}
$$

Another approach is to incorporate the signal on the constraint satisfaction Ψ in the generator loss l_G as a regularization term with weight λ. **[TODO]** Pros and cons?

3.3 Teacher-based training

$$
\begin{aligned}
l_G &= E[D(G(\mathbf{z}))]_{\mathbf{z}\sim P_\mathbf{z}} + \lambda E[D_{KL}(T(G(\mathbf{z})), G(\mathbf{z})]_{\mathbf{z}\sim P_\mathbf{z}} \\
l_D &= E[D(\mathbf{x})]_{\mathbf{x}\sim P_\mathbf{x}} + E[1 - D(G(\mathbf{z}))]_{\mathbf{z}\sim P_\mathbf{z}} \\
l_T &= E[\Psi(T(G(\mathbf{z})))]_{\mathbf{z}\sim P_\mathbf{z}}
\end{aligned}
$$

References

[1] Kuzman Ganchev, João Graça, Jennifer Gillenwater, and Ben Taskar. Posterior regularization for structured latent variable models. *J. Mach. Learn. Res.*, 11:2001–2049, August 2010.

[2] R. Hjelm, A. Jacob, T. Che, K. Cho, and Y. Bengio. Boundary-seeking generative adversarial networks. In *preprint arXiv:1702.08431*.

[3] Zhiting Hu, Xuezhe Ma, Zhengzhong Liu, Eduard H. Hovy, and Eric P. Xing. Harnessing deep neural networks with logic rules. In *ACL (1)*. The Association for Computer Linguistics, 2016.

A Gradient of $D_{KL}(\tilde{p}(\mathbf{x},\mathbf{z})||p_g(\mathbf{x},\mathbf{z}))$

Let θ be the parameters of $g(\mathbf{x}|\mathbf{z})$. Let's assume $\tilde{p}(\mathbf{x},\mathbf{z})$ is constant with respect to θ, i.e. we use the θ from the previous iteration and do not propagate their gradient inside $\tilde{p}$.

$$\begin{aligned}
\nabla_\theta D_{KL}(\tilde{p}(\mathbf{x},\mathbf{z})||p_g(\mathbf{x},\mathbf{z})) &= \nabla_\theta \sum_{\mathbf{z}}\sum_{\mathbf{x}} \tilde{p}(\mathbf{x},\mathbf{z}) \log \frac{\tilde{p}(\mathbf{x},\mathbf{z})}{p_g(\mathbf{x},\mathbf{z})} \\
&= -\sum_{\mathbf{z}}\sum_{\mathbf{x}} \frac{1}{Z_{|\mathbf{z}}} \frac{D(\mathbf{x})}{1-D(\mathbf{x})} g(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) \nabla_\theta \log g(\mathbf{x}|\mathbf{z}) \\
&= -E\left[\sum_{\mathbf{x}} \frac{1}{Z_{|\mathbf{z}}} \frac{D(\mathbf{x})}{1-D(\mathbf{x})} g(\mathbf{x}|\mathbf{z}) \nabla_\theta \log g(\mathbf{x}|\mathbf{z})\right]_{\mathbf{z}\sim p(\mathbf{z})} \\
&\approx -E\left[\sum_{m} \frac{1}{M} \frac{1}{Z_{|\mathbf{z}}} \frac{D(\mathbf{x}^{(m)})}{1-D(\mathbf{x}^{(m)})} \nabla_\theta \log g(\mathbf{x}^{(m)}|\mathbf{z})\right]_{\mathbf{z}\sim p(\mathbf{z})}
\end{aligned}$$

where we approximated the gradient using M samples $\mathbf{x}^{(m)} \sim g(\mathbf{x}|\mathbf{z})$ for each $\mathbf{z}$ ($g(\mathbf{x}|\mathbf{z})$ computes a probability for each component of $\mathbf{x}$, e.g. sigmoid for binary variables, and selecting $\mathbf{x}^{(m)}$ amounts at sampling the discrete values from that probability). The samples can also be used to approximate $Z_{|\mathbf{z}}$ as:

$$Z_{|\mathbf{z}} = \sum_{\mathbf{x}} \frac{D(\mathbf{x})}{1-D(\mathbf{x})} g(\mathbf{x}|\mathbf{z}) \approx \sum_{m} \frac{1}{M} \frac{D(\mathbf{x}^{(m)})}{1-D(\mathbf{x}^{(m)})}$$

Alternatively, one can use $D_{KL}(p_g(\mathbf{x},\mathbf{z})||\tilde{p}(\mathbf{x},\mathbf{z}))$:

$$\begin{aligned}
\nabla_\theta D_{KL}(p_g(\mathbf{x},\mathbf{z})||\tilde{p}(\mathbf{x},\mathbf{z})) &= \nabla_\theta \sum_{\mathbf{z}}\sum_{\mathbf{x}} g(\mathbf{x}|\mathbf{z}) p(\mathbf{z}) \log \frac{p_g(\mathbf{x},\mathbf{z})}{\tilde{p}(\mathbf{x},\mathbf{z})} \\
&= \sum_{\mathbf{z}}\sum_{\mathbf{x}} p(\mathbf{z}) \left[\nabla_\theta g(\mathbf{x}|\mathbf{z}) \log \frac{p_g(\mathbf{x},\mathbf{z})}{\tilde{p}(\mathbf{x},\mathbf{z})} + g(\mathbf{x}|\mathbf{z}) \nabla_\theta \log g(\mathbf{x}|\mathbf{z})\right] \\
&= \sum_{\mathbf{z}}\sum_{\mathbf{x}} p(\mathbf{z}) g(\mathbf{x}|\mathbf{z}) \nabla_\theta \log g(\mathbf{x}|\mathbf{z}) \left[\log \frac{\cancel{g(\mathbf{x}|\mathbf{z})p(\mathbf{z})}}{\frac{1}{Z_{|\mathbf{z}}}\frac{D(\mathbf{x})}{1-D(\mathbf{x})}\cancel{g(\mathbf{x}|\mathbf{z})p(\mathbf{z})}} + 1\right] \\
&= \sum_{\mathbf{z}}\sum_{\mathbf{x}} p(\mathbf{z}) g(\mathbf{x}|\mathbf{z}) \nabla_\theta \log g(\mathbf{x}|\mathbf{z}) \left[\log Z_{|\mathbf{z}} - \log \frac{D(\mathbf{x})}{1-D(\mathbf{x})} + 1\right]
\end{aligned}$$

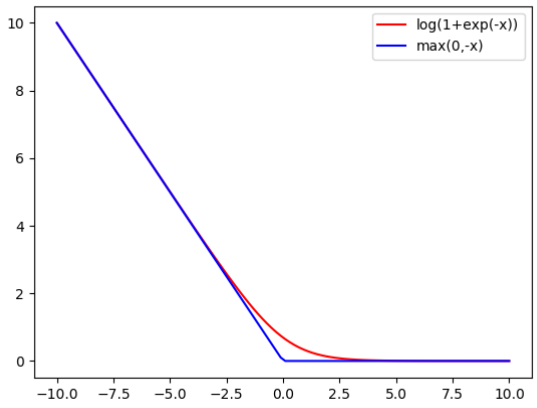

Figure 1: Approximated log gradient

where we used the fact that $\nabla_\theta g(\mathbf{x}|\mathbf{z}) = g(\mathbf{x}|\mathbf{z})\nabla_\theta \log g(\mathbf{x}|\mathbf{z})$

B $\nabla_\theta \log g(\mathbf{x}^{(m)}|\mathbf{z})$ for binary variables

Note that in practice, in [2] $\nabla_\theta \log g(\mathbf{x}^{(m)}|\mathbf{z})$ for binary variables is approximated as:

$$
\begin{aligned}
\nabla_\theta \log g(\mathbf{x}^{(m)}|\mathbf{z}) &= \nabla_\theta \sum_{i=1}^{|\mathbf{x}^{(m)}|} \mathbb{1}_{[x_i^{(m)}=1]} \log g_i(\mathbf{x}^{(m)}|\mathbf{z}) + \mathbb{1}_{[x_i^{(m)}=0]} \log\left(1 - g_i(\mathbf{x}^{(m)}|\mathbf{z})\right) \\
&= \nabla_\theta \sum_{i=1}^{|\mathbf{x}^{(m)}|} \mathbb{1}_{[x_i^{(m)}=1]} \log \sigma(h_i(\mathbf{x}^{(m)}|\mathbf{z})) + \mathbb{1}_{[x_i^{(m)}=0]} \log\left(1 - \sigma(h_i(\mathbf{x}^{(m)}|\mathbf{z}))\right) \\
&= \nabla_\theta \sum_{i=1}^{|\mathbf{x}^{(m)}|} \mathbb{1}_{[x_i^{(m)}=1]} \log \frac{1}{1+\exp -h_i(\mathbf{x}^{(m)}|\mathbf{z})} + \mathbb{1}_{[x_i^{(m)}=0]} \log \frac{\exp -h_i(\mathbf{x}^{(m)}|\mathbf{z})}{1+\exp -h_i(\mathbf{x}^{(m)}|\mathbf{z})} \\
&= -\nabla_\theta \sum_{i=1}^{|\mathbf{x}^{(m)}|} \log\left(1+\exp -h_i(\mathbf{x}^{(m)}|\mathbf{z})\right) + \mathbb{1}_{[x_i^{(m)}=0]} h_i(\mathbf{x}^{(m)}|\mathbf{z}) \\
&\approx -\nabla_\theta \sum_{i=1}^{|\mathbf{x}^{(m)}|} \max\left(0, -h_i(\mathbf{x}^{(m)}|\mathbf{z})\right) + \mathbb{1}_{[x_i^{(m)}=0]} h_i(\mathbf{x}^{(m)}|\mathbf{z})
\end{aligned}
$$

where we used the approximation $\log(1+\exp -x) \approx \max(0, -x)$, see Figure 1.

C $\nabla_\theta \log g(\mathbf{x}^{(m)}|\mathbf{z})$ for multinomial variables

$$\begin{aligned}
\nabla_\theta \log g(\mathbf{x}^{(m)}|\mathbf{z}) &= \sum_{i=1}^{|\mathbf{x}^{(m)}|} \sum_{j=1}^{c} \mathbb{1}_{[x_i^{(m)}=j]} \nabla_\theta \log \frac{\exp h_{i,j}(\mathbf{x}^{(m)}|\mathbf{z})}{\sum_{j'=1}^{c} \exp h_{i,j'}(\mathbf{x}^{(m)}|\mathbf{z})} \\
&= \sum_{i=1}^{|\mathbf{x}^{(m)}|} \sum_{j=1}^{c} \mathbb{1}_{[x_i^{(m)}=j]} \left(\nabla_\theta h_{i,j}(\mathbf{x}^{(m)}|\mathbf{z}) - \nabla_\theta \log \sum_{j'=1}^{c} \exp h_{i,j'}(\mathbf{x}^{(m)}|\mathbf{z}) \right) \\
&= \sum_{i=1}^{|\mathbf{x}^{(m)}|} \sum_{j=1}^{c} \mathbb{1}_{[x_i^{(m)}=j]} \left(\nabla_\theta h_{i,j}(\mathbf{x}^{(m)}|\mathbf{z}) - \frac{\sum_{j'=1}^{c} \exp h_{i,j'}(\mathbf{x}^{(m)}|\mathbf{z}) \nabla_\theta h_{i,j'}(\mathbf{x}^{(m)}|\mathbf{z})}{\sum_{j'=1}^{c} \exp h_{i,j'}(\mathbf{x}^{(m)}|\mathbf{z})} \right)
\end{aligned}$$

where c is the number of possible values of each multinomial variable, $h_{i,j}(\mathbf{x}^{(m)}|\mathbf{z})$ is the output of the j^{th} channel for the i^{th} variable, corresponding to the unnormalized probability of outcome j for that variable.