Distributional Reinforcement Learning for Energy-Based Sequential Models

Tetiana Parshakova
Stanford University[*]
tetianap@stanford.edu

Jean-Marc Andreoli **Marc Dymetman**
Naver Labs Europe
{jean-marc.andreoli,marc.dymetman}@naverlabs.com

Abstract

Global Autoregressive Models (GAMs) are a recent proposal [15] for exploiting global properties of sequences for data-efficient learning of seq2seq models. In the first phase of training, an Energy-Based model (EBM) [10] over sequences is derived. This EBM has high representational power, but is unnormalized and cannot be directly exploited for sampling. To address this issue [15] proposes a distillation technique, which can only be applied under limited conditions. By relating this problem to Policy Gradient techniques in RL, but in a *distributional* rather than *optimization* perspective, we propose a general approach applicable to any sequential EBM. Its effectiveness is illustrated on GAM-based experiments.

1 Introduction

The mainstream autoregressive sequence models [6, 22, 5, 24]) form a subclass of sequential energy-based models (sequential EBMs) [10]. While the former are locally normalized and easy to train and sample from, the latter allow global constraints, greater expressivity, and potentially better sample efficiency, but lead to unnormalized distributions and are more difficult to use for inference and evaluation. We exploit a recently introduced class of energy-based models, Global Autoregressive Models (GAMs) [15], which combine a locally normalized component (that is, a first, standard, autoregressive model, denoted r) with a global component and use these to explore some core research questions about sequential EBMs, focussing our experiments on synthetic data for which we can directly control experimental conditions. We dissociate the (relatively easy) task of learning from the available data an energy-based *representation* (Training-1), from the more challenging task of *exploiting* that representation to produce samples or evaluations (Training-2).

In this paper, we provide a short self-contained introduction to GAMs and to their two-stage training procedure. However our main focus is about Training-2. For that task [15] proposed a Distillation technique to project the Energy-Based representation (denoted by P_λ) obtained at the end of Training-1 into a final autoregressive model (denoted π_θ), with better test perplexity than the initial r, but this technique was limited to cases where it was possible to sample from P_λ at training time. One key observation of the current submission is that Training-2, considered as the general problem of deriving an autoregressive model from an energy-based model (not necessarily obtained through Training-1) has strong similarities with the training of policies in Reinforcement Learning (RL), but in a *distributional* rather than in an *optimization* perspective as in standard RL. We then propose a distributional variant of the Policy Gradient technique (Distributional Policy Gradient: DPG) which has wider applicability than distillation. We conduct GAM-based experiments to compare this technique with distillation, in synthetic data conditions where distillation is feasible, and show that DPG works as well as distillation. In both cases, in small data conditions, the policies (aka autoregressive) models obtained at the end of the process are very similar and show strong perplexity reduction over the standard autoregressive models.

[*] Work done while at Naver Labs Europe.

Section 2 provides an overview of GAMs. Section 3 explains the training procedure, with focus on EBMs and relations to RL. Section 4 presents experiments and results. For space reasons we use the Supplementary Material (Sup. Mat.) to provide some details and to discuss related work.

2 Model

2.1 Background

Autoregressive models (AMs) These are currently the standard for neural seq2seq processing, with such representatives as RNN/LSTMs [6, 22], ConvS2S [5], Transformer [24]). Formally, they are defined though a distribution $r_\eta(x|C)$, where x is a target sequence to be generated, and C is a context, with $r_\eta(x|C) \doteq \prod_i s_\eta(x_i|x_1, \ldots, x_{i-1}, C)$, and where each $s_\eta(x_i|x_1, \ldots, x_{i-1}, C)$ is a normalized conditional probability over the next symbol of the sequence, computed by a neural network (NN) with parameters η. The local normalization of the incremental probabilities implies the overall normalization of the distribution $r_\eta(x|C)$. In RL terminology, AMs can also be seen as *policies* where actions are symbols and states are sequence prefixes.

Energy-Based Models (EBMs) EBMs are a generic class of models, characterized by an energy function $U_\eta(x|C)$ computed by a neural network parametrized by η [10]. Equivalently, they can be seen as directly defining a potential (an unnormalized probability distribution) $P_\eta(x|C) = e^{-U_\eta(x|C)}$, and indirectly the normalized distribution $p_\eta(x|C) = 1/Z_\eta(C)\ P_\eta(x|C)$, with $Z_\eta(C) = \sum_x P_\eta(x|C)$. Here we will identify an EBM with its potential (the P_η form) and be concerned exclusively with sequential EBMs, that is, the case where x is a sequence.

2.2 GAMs

We employ a specific class of sequential EBMs, *Global Autoregressive Models* (GAMs), which we summarize here (for details please see [15]). GAMs exploit both local autoregressive properties as well as global properties of the sequence x. A GAM is an unnormalized potential $P_\eta(x|C)$ over x, parametrized by a vector $\eta = \eta_1 \oplus \eta_2$, which is the product of two factors:

$$P_\eta(x|C) = r_{\eta_1}(x|C) \cdot e^{\langle \lambda_{\eta_2}(C),\ \phi(x;C)\rangle}. \tag{1}$$

Here the factor $r_{\eta_1}(x|C)$ is an autoregressive model for generating x in the context C, parametrized by η_1. The factor $e^{\langle \lambda_{\eta_2}(C),\ \phi(x;C)\rangle}$ on the other hand, is a *log-linear* potential [8], where $\phi(x;C)$ is a vector of predefined real features of the pair (x, C), which is combined by a scalar product with a real vector $\lambda_{\eta_2}(C)$ of the same dimension, computed by a network parametrized by η_2. The normalized distribution associated with the GAM is $p_\eta(x|C) = \frac{P_\eta(x|C)}{Z_\eta(C)}$, where $Z_\eta(C) = \sum_x P_\eta(x|C)$.

The motivations for GAMs are as follows. The first factor guarantees that the GAM will have at least the same effectiveness as standard autoregressive models to model the local, incremental, aspects of sequential data.The second factor can be seen as providing a "modulation" on the first one. While we could have chosen any energy-based potential for that factor, the log-linear form has several advantages. First, the features $\phi(x;C)$ provide prior knowledge to the model by drawing its attention to potentially useful global sequence properties that may be difficult for the AM component to discover on its own. Second, log-linear models enjoy the following important property: at maximum likelihood, the features expectations according to the model and to the data are equal ("moment matching" property).

In our experiments, we focus on a simple unconditional (language modelling) version of GAMs, of the form:

$$P_\lambda(x) \doteq r(x) \cdot e^{\langle \lambda,\ \phi(x)\rangle}, \tag{2}$$

where the autoregressive factor $r = r_{\eta_1}$ is first learnt on the training dataset of sequences D and then kept fixed, and where the parameter vector λ is then trained on top of r, also on D. We denote by $p_\lambda(x)$ the normalized distribution associated with $P_\lambda(x)$.

3 Training

We assume that we are given a training data set D (resp. a validation set V, a test set T) of sequences x, and a finite collection of real-valued feature functions $\phi_1, \ldots, \phi_k$. The GAM training procedure then is performed in two stages (see Fig. 1).

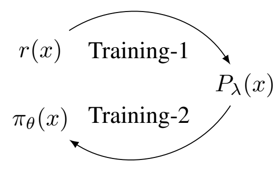

Figure 1: Two-stage training. At the end of the process, we compare the perplexities of r and π_θ on test data: $CE(T, r)$ vs. $CE(T, \pi_\theta)$.

3.1 Training-1: from data to energy-based representation

This phase consists in training P_λ by max-likelihood (ML) on D. We start by training an AM $r = r_{\eta_1}$ (our initial policy) on D, in the standard way. We then fit the log-linear weight vector λ to the data. In order to do that, we denote by $\log p_\lambda(D)$ the log-likelihood of the data, and perform SGD over λ by observing that (2) implies:

$$\nabla_\lambda \log p_\lambda(D) = |D| \cdot [E_{x \sim p_D(x)}\ \phi(x) - E_{x \sim p_\lambda(\cdot)}\ \phi(x)], \tag{3}$$

where $E_{x \sim p_D(x)}\ \phi(x)$ (resp. $E_{x \sim p_\lambda(\cdot)}\ \phi(x)$) denotes the expectation (aka moment) of the feature vector relative to the data (resp. to the model). The first moment can be directly computed from the data, but the second moment requires more effort. The most direct way for estimating $E_{x \sim p_\lambda(\cdot)}\ \phi(x)$ would be to produce a random sample from $p_\lambda(\cdot)$ and to compute the mean of $\phi(x)$ over this sample. In general, when starting from an unnormalized P_λ as here, obtaining samples from p_λ can be difficult. One approach consists in applying a Monte-Carlo sampling technique, such as Rejection Sampling (*rs*) [18], and this is one of two techniques that can be applied in the experimental conditions both of [15] and of this paper. However rejection sampling is feasible only in situations where reasonable upper-bounds of the ratio $P(x)/q(x)$ (for q a proposal distribution) can be derived.[2] This is why [15] proposes another technique of wider applicability, Self-Normalized Importance Sampling (*snis*) [14, 26].This technique directly estimates the expectation $E_{x \sim p_\lambda(\cdot)}\ \phi(x)$ without requiring samples from p_λ.

3.2 Training-2: from energy-based representation to distributional policy

The output of the previous stage is an unnormalized EBM, which allows us to compute the potential $P(x) = P_\lambda(x)$ of any given x, but not directly to compute the partition function $Z = \sum_x P(x)$ nor the normalized distribution $p(x) = 1/Z\ P(x) = p_\lambda(x)$ or to sample from it.[3] In RL terms, the score $P(x)$ can be seen as a *reward*. The standard RL-as-optimization view would lead us to search for a way to maximize the expectation of this reward, in other words for a policy π_{θ^*} with $\theta^* = \text{argmax}_\theta\, \mathbb{E}_{x \sim \pi_\theta(\cdot)}\, P(x)$, which would tend to concentrate all its mass on a few sequences.

By contrast, our RL-as-sampling (distributional) view consists in trying to find a policy π_{θ^*} that approximates the distribution p as closely as possible, in terms of cross-entropy CE. We are thus trying to solve $\theta^* = \text{argmin}_\theta\, \text{CE}(p, \pi_\theta)$, with $CE(p, \pi_\theta) = -\sum_x p(x) \log \pi_\theta(x)$. We have:

$$\nabla_\theta\, \text{CE}(p, \pi_\theta) = -\sum_x p(x)\, \nabla_\theta \log \pi_\theta(x) = -\,\mathbb{E}_{x \sim p(\cdot)}\, \nabla_\theta \log \pi_\theta(x). \tag{4}$$

We can apply (4) for SGD optimization, using different approaches.

The simplest approach, **Distillation**, can be employed in situations where we are able to draw, in reasonable training time, a large number of samples $x_1, \ldots, x_K$ from p. We can then exploit (4) directly to update θ, which is in fact equivalent to performing a standard supervised log-likelihood SGD training on the set $\{x_1, \ldots, x_K\}$. This is the approach to Training-2 taken in [15], using rejection sampling at training time for obtaining the samples, and then training θ on these samples to obtain a final AM π_θ which can be used for efficient sampling at test time and for evaluation. The advantage of this approach is that supervised training of this sort is very succesful for standard autoregressive models, with good stability and convergence properties, and an efficient use of the training data through epoch iteration.[4] However, the big disadvantage is its limited applicability, due to restrictive conditions for rejection sampling, as explained earlier.

[2]More sophisticated MCMC sampling techniques with broader applicability exist [18], but they are typically difficult to control and slow to converge.

[3]In our discussion of Training-2, to stress the generality of the techniques employed, we will use $P(x)$ to denote *any* EBM potential over sequences, and $p(x) = 1/Z\ P(x)$, with $Z = \sum_x P(x)$, to denote the associated normalized distribution. Whether $P(x)$ is obtained or not through Training-1 in a GAM-style approach is irrelevant to this discussion.

[4]Epoch iteration might actually be seen as a form of "experience replay", to borrow RL terminology [11].

A central contribution of the present paper is to propose another class of approaches, which does not involve sampling from p, and which relates to standard techniques in RL. We can rewrite the last formula of (4) as:

$$\sum_x p(x)\,\nabla_\theta \log \pi_\theta(x) = \frac{1}{Z}\,\mathbb{E}_{x\sim\pi_\theta(\cdot)}\,\frac{P(x)}{\pi_\theta(x)}\nabla_\theta \log \pi_\theta(x). \tag{5}$$

This formula is very close to the vanilla formulation (aka REINFORCE [25]), we have a reward $R(x)$ and we try to maximize the expectation $\mathbb{E}_{x\sim\pi_\theta(\cdot)}\,R(x)$. It can be shown [23] that $\nabla_\theta\,\mathbb{E}_{x\sim\pi_\theta(\cdot)}\,R(x) = \mathbb{E}_{x\sim\pi_\theta(\cdot)}\,R(x)\,\nabla_\theta \log \pi_\theta(x)$. Thus, in the RL case, an SGD step consists in sampling x from π_θ and computing $R(x)\nabla_\theta \log \pi_\theta(x)$, while the SGD step in (5) only differs by replacing $R(x)$ by $\frac{P(x)}{\pi_\theta(x)}$.[5] We will refer to the approach (5) through the name **Distributional Policy Gradient** (on-policy version) or DPG$_{\text{on}}$ ("on-policy" because the sampling is done according to the same policy π_θ that is being learnt).

An off-policy variant DPG$_{\text{off}}$ of (5) is also possible. Here we assume that we are given some fixed proposal distribution q and we write:

$$\sum_x p(x)\,\nabla_\theta \log \pi_\theta(x) = \frac{1}{Z}\,\mathbb{E}_{x\sim q(\cdot)}\,\frac{P(x)}{q(x)}\nabla_\theta \log \pi_\theta(x). \tag{6}$$

Here the sampling policy q is different from the policy being learnt, and the formula (6) represents a form of Importance Sampling, with q the proposal, typically chosen to be an approximation to p.

We did some initial experiments with DPG$_{\text{on}}$, but found that the method had difficulty converging, probably due in part to the instability induced by the constant change of sampling distribution (namely π_θ). A similar phenomenon is well documented in the case of the vanilla Policy Gradient in standard RL, and techniques such as TRPO [20] or PPO [21] have been developed to control the rate of change of the sampling distribution. In order to avoid such instability, we decided to focus on DPG$_{\text{off}}$, based on Algorithm 1 below.

In this algorithm, we suppose that we have as input a potential function P, and an initial proposal distribution q; in the case of GAMs, we take $P = P_\lambda$ and a good π_{θ_0} is provided by r. We then iterate the collection of episodes x sampled with the *same* q (line 4), and perform SGD updates (line 5) according to (6) ($\alpha^{(\theta)}$ is the learning rate). We do update the proposal q at certain times (line 7), but only based on the condition that the current π_θ is superior to q in terms of perplexity measured on the validation set V, thus ensuring a certain stability of the proposal.

Algorithm 1 DPG$_{\text{off}}$

```
Input: $P$, initial policy $q$
1: $\pi_\theta \leftarrow q$
2: for each iteration do
3:     for each episode do
4:         sample $x$ from $q(\cdot)$
5:         $\theta \leftarrow \theta + \alpha^{(\theta)} \frac{P(x)}{q(x)} \nabla_\theta \log \pi_\theta(x)$
6:     if $\pi_\theta$ is superior to $q$ then
7:         $q \leftarrow \pi_\theta$
Output: $\pi_\theta$
```

This algorithm worked much better than the DPG$_{\text{on}}$ version, and we retained it as our implementation of DPG in all our experiments.

4 Experiments

In order to assess the validity of our approach, we perform experiments under controllable conditions based on synthetic binary sequences. Our setup is similar to that of [15]. We generate datasets D, V, T of binary sequences according to a underlying process p_{true}. This process produces random "white noise" binary strings with fixed length $n = 30$ that are filtered according to whether they contain a specific, fixed, substring ("motif") anywhere inside the sequence. The interest of such a process is that one can efficiently generate datasets (by implementing the filtering process through a probabilistic finite-state automaton) and also directly compute the theoretical entropy (perplexity) of the process (see [15]). Also, [15] observed that $p_{true}(x)$ could be well approximated by a standard autoregressive model $r(x)$ when the training dataset was large.

In these experiments, we employed a GAM architecture according to (2), using a fixed set of five binary features[6]: one feature corresponding to the presence/absence of the motif in the candidate sequence, and four "distractor" features with no (or little) predictive value for the validity of the

[5]The constant factor $1/Z$ can be ignored here: during SGD, it has the effect of rescaling the learning rate.

[6]We also did experiments involving two continuous features (M and v) assessing length, see A.4 in Sup. Mat.

candidate sequence (this feature set, using [15] notation, is denoted in the figures by the mask $ft = 1001111$). We vary the motifs m used, the size of the training set D, and the seeds employed.

Our implementation is based on PyTorch [16], with policies (i.e. autoregressive models r and π_θ) implemented as LSTM models over the vocabulary $\{0, 1, \langle \text{EOS} \rangle\}$, with each token represented as a one-hot vector.

The specific experimental setup that we use, due to the nature of the features (binary features or length features M, v), permits to perform Training-2 through distillation (the method used in [15]). In these experiments, we want to confirm that the more generally applicable DPG method works equally well. We do so by varying the training dataset size D and by computing the test perplexity (cross-entropy) of the π_θ obtained at the end of Training-1 + Training-2, and then checking that both distillation and DPG lower this perplexity relative to that of the initial r, under small data conditions (data efficiency). But we also confirm that in Training-2, both distillation and DPG are able to almost perfectly approximate the EBM P_λ obtained at the end of Training-1 (that is, to approximate the associated normalized p_λ); in other words, when P_λ is able to model the p_{true} accurately (which depends on both the quality of the initial r and on the ability of the features to fit the underlying process), then DPG is able to produce a π_θ that accurately represents p_{true}.

Overall Training: Distillation vs. DPG We consider a situation where Training-1 is done through *snis*, but Training-2 is done either through Distillation or through DPG (i.e. DPG$_{\text{off}}$). Figure 2 illustrates this case. Here the motif, feature vector, and seed are fixed, but the training size $|D|$ varies from 500 to $2 \cdot 10^4$) (the size of the test set T is fixed at $5 \cdot 10^3$).

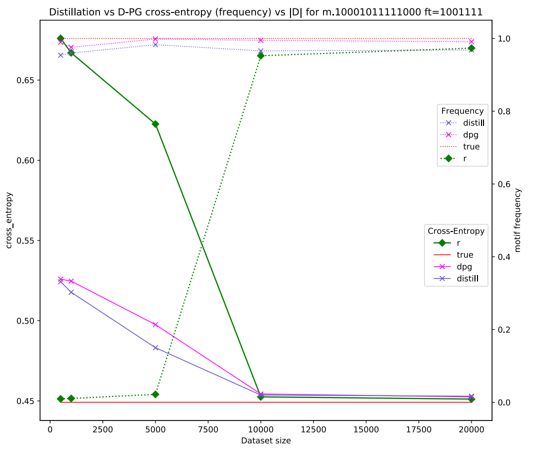

Figure 2: Distillation vs. DPG

The solid lines represent the cross entropies of the final π_θ relative to the test set, with the scale located on the left side of the figure, while the dashed lines are the frequencies of the motif m (computed on 2000 strings sampled from π_θ) with the corresponding scale on the right. We distinguish two versions of Training-2, one based on distillation (`distill`), the other on DPG (`dpg`).

First consider the points above $|D| = 5000$, and the solid lines: for both `distill` and `dpg`, we have $CE(T, r) \gg CE(T, \pi_\theta) \approx H(p_{true})$: π_θ is more data efficient than the initial AM r. For smaller data conditions, the tendency is even stronger, while larger D lead to an initial r which is already very good, and on which the two-stage training cannot improve.

Similar conclusions hold for the motif frequencies of π_θ compared to r: in small data conditions, the motif is much more frequently present when using π_θ.

Finally, comparing `distill` and `dpg`, we see that the performances are very comparable, in this case with a slight advantage of `distill` over `dpg` in perplexities but the reverse in motif frequencies.

Effectiveness of DPG in approximating p To emphasize the performance of DPG in Training-2 (that is, its effectiveness at finding a distributional policy π_θ for an EBM representation $P(x)$), independently of the quality of Training-1), we considered two alternatives for P. The first one took $P = P_\lambda$, the energy-based model obtained from Training-1. In our specific experimental conditions, we were able to accurately estimate (via importance sampling) the partition function Z and therefore to compute the cross entropy $CE(T, p_\lambda)$, and to compare it with $CE(T, \pi_\theta)$: they were extremely close. We confirmed that finding by considering an alternative where P was defined *a priori* in such a way that we could compute p and $CE(T, p)$ exactly, observing the same behavior. Details are provided in Sup. Mat. A.3.

Results In Table 1 we compute the means of ratios of different quantities across experiments with different motifs, features and seeds: $motif \in \{1000101000101, 1011100111001,$

Table 1: Statistics over: $motif \in \{1000101000101, 1011100111001,\ 10001011111000\}, ft \in \{1001111, Mv1001111\}$, $seed \in \{1234, 4444\}$.

height$\|D\|$	$\frac{\text{CE}(T,\pi_\theta^{dpg})}{\text{CE}(T,\pi_\theta^{dis})}$	$\frac{\text{mtf\_frq}(\pi_\theta^{dpg})}{\text{mtf\_frq}(\pi_\theta^{dis})}$	$\frac{\text{CE}(T,\pi_\theta^{dpg})}{\text{CE}(T,r)}$	$\frac{\text{CE}(T,\pi_\theta^{dpg})}{H(p_{true})}$	$\frac{\text{mtf\_frq}(\pi_\theta^{dpg})}{\text{mtf\_frq}(r)}$	$\frac{\text{CE}(T,\pi_\theta^{dis})}{\text{CE}(T,r)}$	$\frac{\text{mtf\_frq}(\pi_\theta^{dis})}{\text{mtf\_frq}(r)}$
500	1.008	1.252	0.76	1.18	281.51	0.758	224.94
1000	1.014	1.102	0.762	1.178	240.40	0.76	218.24
5000	1.019	1.21	0.865	1.059	34.73	0.847	28.69
10000	1.014	1.067	0.968	1.023	2.17	0.963	2.04
20000	1.004	1.023	1.0	1.006	1.03	1.002	1.01

$10001011111000\}, ft \in \{1001111, Mv1001111\}$, $seed \in \{1234, 4444\}$. In all cases Training-1 is performed using *snis*.

These statistics confirm the tendencies illustrated in the previous plots. Namely, when $|D|$ increases the test cross entropy $CE(T, \pi_\theta)$ gets closer to the theoretical one $H(p_{true})$. Also π_θ outperforms r in small conditions of $|D|$ for the two modes of Training-2: the columns $\frac{\text{CE}(T,\pi_\theta^{dpg})}{\text{CE}(T,r)}$ and $\frac{\text{CE}(T,\pi_\theta^{dis})}{\text{CE}(T,r)}$ show that the models approximate the true process more closely than the initial r in settings with $|D| < 10^4$. Similar conclusions can be drawn when comparing the motif frequencies of π_θ and r. Further, according to data in columns $\frac{\text{CE}(T,\pi_\theta^{dpg})}{\text{CE}(T,\pi_\theta^{dis})}$ and $\frac{\text{mtf\_frq}(\pi_\theta^{dpg})}{\text{mtf\_frq}(\pi_\theta^{dis})}$, we see that DPG and distillation have comparable efficiency for obtaining the final policy. DPG gives rise to a policy that has better motif frequency but slightly worse cross-entropy than the one from distillation.

5 Conclusion

Motivated by the GAM formalism for learning sequential models,[7] we proposed some RL-inspired techniques for obtaining distributional policies approximating the normalized distribution associated with an energy-based model over sequences. We took some first experimental steps, in controlled synthetic conditions, for confirming that these techniques were working.

While the main algorithm (DPG$_{\text{off}}$) proposed here for computing distributional policies is generic in the sense that it only requires a potential $P(x)$ and a proposal q, the fact that GAMs intrinsically enclose an autoregressive policy r that can be used to initialize such a proposal is an important advantage. It should also be observed that the division of work in GAMs between Training-1 and Training-2 helps clarifying a distinction that should be made about training sequential EBMs from data. [15] already observed that training the representation P_λ could be much easier than extracting an autoregressive model from it.[8] If we think in the terms of the current paper, we can further observe that while Training-2 has direct connections to RL (exploiting a given reward to obtain a policy), Training-1 has some similarities to *Inverse RL* [19, 12]: deriving a reward *from the training data*, here purely inside a max-likelihood approach. Trying to combine the two aspects in one direct algorithm would only blur the true nature of the problem.

The move from the standard optimization view of RL and the sampling (aka distributional) view advocated here is a natural one. Optimization can be seen as an extreme case of sampling with a low temperature, and the approach to distributional policies developped in our Algorithm 1 might be a way for developing stable algorithms for standard RL purposes (a related approach is proposed in [13]).

Our importation of policy gradient from standard RL to the distributional view only scratches the surface, and another promising line of research would be to adapt methods for local credit assignment, such as actor-critic techniques, to the problem of sampling from an energy-based model.

Acknowledgements Thanks to Tomi Silander and Hady Elsahar for discussions and feedback.

[7]The limitation to *sequential* EBMs is not as serious as it seems. Many objects can be decomposed into sequences of actions, and EBMs over such objects could then be handled in similar ways to those proposed here.

[8]There are some extreme situations where the P_λ obtained at the end of Training-1 can perfectly represent the true underlying process, but no policy has a chance to approximate p_λ. This can happen with features associated with complex filters (e.g. of a cryptographic nature) used for generating the data, which can be easily detected as useful during Training-1, but cannot feasibly be projected back onto incremental policies.

References

[1] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2442–2452, Berlin, Germany, August 2016. Association for Computational Linguistics.

[2] Dzmitry Bahdanau, Philemon Brakel, Kelvin Xu, Anirudh Goyal, Ryan Lowe, Joelle Pineau, Aaron Courville, and Yoshua Bengio. An Actor-Critic Algorithm for Sequence Prediction. (2015):1–17, 2016.

[3] David Belanger and Andrew McCallum. Structured prediction energy networks. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 983–992. JMLR.org, 2016.

[4] Marc G. Bellemare, Will Dabney, and Rémi Munos. A Distributional Perspective on Reinforcement Learning. *arXiv:1707.06887 [cs, stat]*, July 2017. arXiv: 1707.06887.

[5] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional sequence to sequence learning. *CoRR*, 2017. cite arxiv:1705.03122.

[6] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[7] Natasha Jaques, Shixiang Gu, Dzmitry Bahdanau, Jose Miguel Hernandez Lobato, Richard E. Turner, and Doug Eck. Tuning recurrent neural networks with reinforcement learning. 2017.

[8] Tony Jebara. Log-Linear Models, Logistic Regression and Conditional Random Fields, 2013.

[9] Taesup Kim and Yoshua Bengio. Deep directed generative models with energy-based probability estimation. *CoRR*, abs/1606.03439, 2016.

[10] Yann LeCun, Sumit Chopra, Raia Hadsell, Marc'Aurelio Ranzato, and Fu Jie Huang. A Tutorial on Energy-Based Learning. *Predicting Structured Data*, pages 191–246, 2006.

[11] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[12] Andrew Y. Ng and Stuart J. Russell. Algorithms for inverse reinforcement learning. In *Proceedings of the Seventeenth International Conference on Machine Learning*, ICML '00, pages 663–670, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[13] Mohammad Norouzi, Samy Bengio, Zhifeng Chen, Navdeep Jaitly, Mike Schuster, Yonghui Wu, and Dale Schuurmans. Reward augmented maximum likelihood for neural structured prediction. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16, pages 1731–1739, USA, 2016. Curran Associates Inc.

[14] Art Owen. Adaptive Importance Sampling (slides). 2017.

[15] Tetiana Parshakova, Jean-Marc Andreoli, and Marc Dymetman. Global Autoregressive Models for Data-Efficient Sequence Learning. In *CoNLL 2019*, Hong Kong, November 2019.

[16] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

[17] Marc'Aurelio Ranzato, Sumit Chopra, Michael Auli, and Wojciech Zaremba. Sequence level training with recurrent neural networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[18] Christian P. Robert and George Casella. *Monte Carlo Statistical Methods (Springer Texts in Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2005.

[19] Stuart Russell. Learning agents for uncertain environments (extended abstract). In *Proceedings of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 101–103, New York, NY, USA, 1998. ACM.

[20] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897, 2015.

[21] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint: 1707.06347*, 2017.

[22] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014.

[23] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018.

[24] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6000–6010, 2017.

[25] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. In *Machine Learning*, pages 229–256, 1992.

[26] Y. Bengio and J. S. Senecal. Adaptive Importance Sampling to Accelerate Training of a Neural Probabilistic Language Model. *Ieee Transactions on Neural Networks*, 19(4):713–722, 2008.