ADACAT: Adaptive Categorical Discretization for Autoregressive Models (Supplementary material)

Qiyang Li[1] **Ajay Jain**[1] **Pieter Abbeel**[1]

[1]University of California Berkeley, Berkeley, CA, USA

A WHY IS TRAINING WITH THE NON-SMOOTHED LOSS UNSTABLE?

We hypothesize that the training instability comes from the biased gradients from the non-smoothed loss. We demonstrate where this bias could come from in a simple 1-D example below by analyzing the gradient of the negative log-likelihood (NLL) and the gradient of the Monte-Carlo approximation of the NLL:

$$\nabla_\theta \mathcal{L}_{\text{ll}} = -\nabla_\theta \int_x p_{\text{data}}(x) \log p_\theta(x) dx$$
$$\nabla_\theta \hat{\mathcal{L}}_{\text{ll}} = -\frac{1}{n} \sum_{d=1}^{n} \nabla_\theta \log p_\theta(x_d)$$

The first gradient $\nabla_\theta \mathcal{L}_{\text{ll}}$ is the correct gradient that we hope to approximate with $\nabla_\theta \hat{\mathcal{L}}_{\text{ll}}$ during training. However, due to the non-differentiability of $p_\theta(x_d)$ with respect to θ at the boundary of each uniform mixture component, the following statement is *not* generally true:

$$\nabla_\theta \mathcal{L}_{\text{ll}} = \mathbb{E}_{x_1, \cdots, x_n \overset{i.i.d.}{\sim} p_{\text{data}}} \left[\nabla_\theta \hat{\mathcal{L}}_{\text{ll}} \right]$$

We provide a simple example where the gradient approximation is biased. Let p_{data} be a uniform distribution in $[0., 1.)$ and p_θ be parameterized by the AdaCat distribution with $w_1 = w_2 = 0.5, h_1 = h_2 = 0.5$ (2 components). Recall that $\theta = [\psi_1, \psi_2, \phi_1, \phi_2]$, which produces w_1, w_2, h_1, h_2 through softmax. The correct gradient and the expectation of the approximated gradient can be computed analytically as follows:

$$\begin{aligned}
\nabla_\theta \mathcal{L}_{\text{ll}} &= -\nabla_\theta \left[\int_x p_{\text{data}}(x) \log p_\theta(x) dx \right] \\
&= -\nabla_\theta \left[w_1 \log \left(\frac{h_1}{w_1} \right) + w_2 \log \left(\frac{h_2}{w_2} \right) \right] \\
&= -\Bigg[\log \left(\frac{h_1}{w_1} \right) \nabla_\theta w_1 + \log \left(\frac{h_2}{w_2} \right) \nabla_\theta w_2 + \\
&\quad w_1 \nabla_\theta \log \left(\frac{h_1}{w_1} \right) + w_2 \nabla_\theta \log \left(\frac{h_2}{w_2} \right) \Bigg]
\end{aligned}$$

Accepted for the 38th Conference on Uncertainty in Artificial Intelligence (UAI 2022).

$$
\begin{aligned}
\mathbb{E}_{x_{1:n}}\left[\nabla_\theta \hat{\mathcal{L}}_{\text{ll}}\right] &= -\mathbb{E}_x\left[\nabla_\theta \log p_\theta(x)\right] \\
&= -\mathbb{E}_x\Bigg[\mathbb{I}\left\{x < 0.5\right\}\nabla_\theta \log\left(\frac{h_1}{w_1}\right) + \\
&\quad \mathbb{I}\left\{x \geq 0.5\right\}\nabla_\theta \log\left(\frac{h_2}{w_2}\right)\Bigg] \\
&= -\mathbb{E}_x\Bigg[\mathbb{I}\left\{x < 0.5\right\}\Bigg]\nabla_\theta \log\left(\frac{h_1}{w_1}\right) + \\
&\quad \mathbb{E}_x\Bigg[\mathbb{I}\left\{x \geq 0.5\right\}\Bigg]\nabla_\theta \log\left(\frac{h_2}{w_2}\right) \\
&= -w_1\nabla_\theta \log\left(\frac{h_1}{w_1}\right) - w_2\nabla_\theta \log\left(\frac{h_2}{w_2}\right)
\end{aligned}
$$

The correct gradient (first equation) contains two more terms (that involves $\nabla_\theta w_1$ and $\nabla_\theta w_2$) than the expectation of the sample gradient (second equation).

B INTERPRETATION OF THE SMOOTHED OBJECTIVE

The gradient of our smoothed objective is an *unbiased* gradient estimator of the negative log-likelihood of the *smoothed* data distribution under the model. We state the connection in more precise terms (with $m = 1$ for readability):

Claim B.1.

$$
\mathbb{E}_{x_1,\cdots,x_n \overset{i.i.d.}{\sim} p_{\text{data}}}\left[\nabla_\theta \hat{\mathcal{L}}_s\right] = \nabla_\theta \mathbb{E}_{\tilde{x}\sim\tilde{p}_{\text{data}}}\left[\log p_\theta(\tilde{x})\right]
$$

where the smoothed data distribution is defined as

$$
\tilde{p}_{\text{data}}(\tilde{x}) = \int_x \zeta(\tilde{x}|x)p_{\text{data}}(x)dx
$$

Before we move on to the proof, we first check that the smoothed data distribution $\tilde{p}_{\text{data}}(\tilde{x})$ is in fact a valid distribution:

$$
\begin{aligned}
\int_{\tilde{x}} \tilde{p}_{\text{data}}(\tilde{x})d\tilde{x} &= \int_{\tilde{x}}\int_x \zeta(\tilde{x}|x)p_{\text{data}}(x)dxd\tilde{x} \\
&= \int_x \left(\int_{\tilde{x}} \zeta(\tilde{x}|x)d\tilde{x}\right) p_{\text{data}}(x)dx \\
&= \int_x p_{\text{data}}(x)dx \\
&= 1.
\end{aligned}
$$

Proof.

$$
\begin{aligned}
&\mathbb{E}_{x_{1:n} \overset{i.i.d}{\sim} p_{\text{data}}(x)} \left[\nabla_\theta \hat{\mathcal{L}}_s \right] \\
&\quad = \mathbb{E}_{x_{1:n}} \left[\nabla_\theta \left[\frac{1}{n} \sum_{d=1}^{n} \int_{\tilde{x}_d} \zeta(\tilde{x}_d | x_d) \log p_\theta(\tilde{x}_d) d\tilde{x}_d \right] \right] \\
&\quad = \nabla_\theta \mathbb{E}_{x_{1:n}} \left[\frac{1}{n} \sum_{d=1}^{n} \int_{\tilde{x}_d} \zeta(\tilde{x}_d | x_d) \log p_\theta(\tilde{x}_d) d\tilde{x}_d \right] \\
&\quad = \nabla_\theta \mathbb{E}_{x \sim p_{\text{data}}(x)} \left[\int_{\tilde{x}} \zeta(\tilde{x} | x) \log p_\theta(\tilde{x}) d\tilde{x} \right] \\
&\quad = \nabla_\theta \int_x \int_{\tilde{x}} p_{\text{data}}(x) \zeta(\tilde{x} | x) \log p_\theta(\tilde{x}) d\tilde{x} dx \\
&\quad = \nabla_\theta \int_{\tilde{x}} \left[\int_x p_{\text{data}}(x) \zeta(\tilde{x} | x) dx \right] \log p_\theta(\tilde{x}) d\tilde{x} \\
&\quad = \nabla_\theta \int_{\tilde{x}} \tilde{p}_{\text{data}}(\tilde{x}) \log p_\theta(\tilde{x}) d\tilde{x} \\
&\quad = \nabla_\theta \mathbb{E}_{\tilde{x} \sim \tilde{p}_{\text{data}}} \left[\log p_\theta(\tilde{x}) \right]
\end{aligned}
$$

□

We would like to emphasize that the reason we could move the ∇_θ operator outside the expectation is because the integral $\int_{\tilde{x}} \zeta(\tilde{x}|x) \log p_\theta(\tilde{x}) d\tilde{x}$ is differentiable with respect to θ: we analytically evaluate the integral using the cumulative density function (Equation 8). This is in contrast to the non-smoothed objective, which uses the discontinuous probability density function as analyzed in the previous section. Furthermore, as long as the smoothing kernel ζ has small bandwidth (λ), $\tilde{p}_{\text{data}}$ would be close to p_{data}. As a result, the gradient of the smoothed objective would closely approximate the true objective that needs to be optimized $\mathbb{E}_{x \sim p_{\text{data}}}[\log p_\theta(x)]$.

C EXPERIMENT DETAILS

To complete all the experiments we did in this paper, we approximatedly used 200 hours of machine time on NVIDIA DGX-1 machine with Tesla V100. The machine has 8 Tesla V100 GPUs and 2x 20-core Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz (see more detailed specs here `https://images.nvidia.com/content/pdf/dgx1-v100-system-architecture-whitepaper.pdf`). We also provide additional details of our hyperparameters for each task below.

C.1 TABULAR DATASETS

We provide the parameters used for ADACAT and the uniform baseline in Table 1 and 2. Note that the parameters are different because they are individually tuned for each method via a grid search. Other shared hyperparameters are provided in Table 3.

Dataset	b	H	λ	k	n
POWER	32	500	1e-5	100	10000
GAS	32	1000	1e-4	1000	10000
HEPMASS	4	500	1e-4	100	10000
MINIBOONE	32	500	1e-4	100	1000

Table 1: **Hyperparameters of ADACAT for UCI Datasets**. H is the number of hidden neurons used in each four-layer network. b is the number of Fourier feature pairs used, λ is the width of the smoothing distribution, k is the number of mixture components used, and n is the batch size.

Dataset	b	H	λ	k	n
POWER	32	500	1e-5	100	10000
GAS	32	1000	1e-4	200	10000
HEPMASS	4	500	1e-4	300	10000
MINIBOONE	32	500	1e-4	100	1000

Table 2: **Hyperparameters of the Uniform Baseline for UCI Datasets**. H is the number of hidden neurons used in each four-layer network. b is the number of Fourier feature pairs used, λ is the width of the smoothing distribution, k is the number of mixture components used, and n is the batch size.

Hyperparameter	Value
Weight Decay	1e-4
Learning Rate	1e-4 (Halved every 100 gradient steps)
# of Epochs	400
# of Layers	4 (3 hidden layers)
Optimizer	Adam, $(\beta_1, \beta_2, \epsilon) = (0.9, 0.999, 10^{-8})$

Table 3: **Other Shared Hyperparameters for UCI Datasets**

C.2 MNIST

For MNIST image generation experiments, we mostly borrow the default training hyperparameters from the minGPT repo (`github.com/karpathy/minGPT`) except for the batch size and the size of the network (number of layers and number of heads). See Table 4 below.

Hyperparameter	Value
Dropout Rate	0.1
Learning Rate	3e-4
Weight Decay	0.1
Gradient Clipping	0.1
# of Layers	0.1
Optimizer	Adam, $(\beta_1, \beta_2, \epsilon) = (0.9, 0.95, 10^{-8})$
# of Epochs	100
Embedding Size	768
# of Layers	4
# of Att. Heads	4

Table 4: **Hyperparameters for MNIST Experiment**

C.3 OFFLINE RL

We used the same hyperparameters as used in the official trajectory transformer codebase (`github.com/jannerm/trajectory-transformer`). The only differences aside from the embedding layer change lie in the implementation details of trajectory sampling:

Mid-Point Sampling vs. Uniform Sampling The original trajectory transformer codebase implements mid-point sampling where a bin is first sampled with the corresponding predicted probability. Then, the mid-point of the bin is used as the sampling result. This is different from our implementation where we use a uniform sampling over the selected bin since we no longer treat states and actions as discretized tokens.

Action Sampler and Observation Sampler The original trajectory transformer codebase employs different sampling strategies for actions and observations. In particular, the observation is decoded greedily by always choosing the most probable bin and the action is decoded by sampling from the top 40% bins (`cdf_act=0.6`). We follow the same strategies with the only difference that we use uniform sampling within the bin whereas the original code uses mid-point sampling (see the discussion in the previous paragraph). In addition, we removed the 40% cut-off (`cdf_act=0`) for the `Hopper-Medium` task as we found it to improve the planning performance.

Exponential Moving Average We also planned with the exponential moving average (with a coefficient of 0.995) of the parameters for the `HalfCheetah-Medium` and `Walker2d-Medium` tasks to help reduce the variance of the planning results. We did not use it for the `Hopper-Medium` task because we did not find it to help.

C.4 WAVENET

Our WaveNet experiment code closely follows the implementation of Yamamoto [2019], available at `github.com/r9y9/wavenet_vocoder`. The model is a WaveNet audio decoder based on dilated convolutions and conditioned on ground-truth Mel-spectrograms extracted from the audio files of the LJSpeech 1.1 dataset. Audio is sampled at 22050 Hz. Our μ-law baseline uses a categorical output distribution with an n-bit μ-law quantized waveform as input (8-bit for 256 bins). Waveform samples are represented through one-hot vectors. For Mixture of Logistics (MoL) and Gaussian conditionals, the input waveform is represented as 16-bit raw audio. WaveNet is optimized with Adam at a learning rate of 0.001 that halves every 200K steps. Checkpoints are averaged during training with an exponential moving average, with a decay rate of 0.9999. The EMA model is used for evaluation. For AdaCat conditionals, the model is trained with the uniformly smoothed objective with $\lambda = 0.0001, 0.00005$ or 0.00001.

References

Ryuichi Yamamoto. Wavenet vocoder. `https://github.com/r9y9/wavenet_vocoder`, 2019.