A Supplementary Material

A.1 SGD in Energy-based models

The formula 1 is fundamental for studying the SGD behavior of Energy-based models, and for convenience, we provide a derivation here.

If we define $Z_\eta(C) \doteq \sum_x P_\eta(x|C)$, we find that:

$$\begin{aligned}
\nabla_\eta \log Z_\eta(C) &= \nabla_\eta \log \sum_x P_\eta(x|C) \\
&= \frac{1}{Z_\eta(C)} \nabla_\eta \sum_x P_\eta(x|C) \\
&= \frac{1}{Z_\eta(C)} \sum_x \nabla_\eta P_\eta(x|C) \\
&= \frac{1}{Z_\eta(C)} \sum_x P_\eta(x|C) \nabla_\eta \log P_\eta(x|C) \\
&= \sum_x p_\eta(x|C) \nabla_\eta \log P_\eta(x|C) \\
&= E_{x \sim p_\eta(\cdot|C)} \nabla_\eta \log P_\eta(x|C).
\end{aligned}$$

We then have:

$$\begin{aligned}
\nabla_\eta \log p_\eta(x|C) &= \nabla_\eta \log P_\eta(x|C) - \nabla_\eta \log Z_\eta(C) \\
&= \nabla_\eta \log P_\eta(x|C) \\
&\quad - E_{x \sim p_\eta(\cdot|C)} \nabla_\eta \log P_\eta(x|C).
\end{aligned}$$

A.2 The relevance of finite state automata; connections and differences with Reinforcement Learning

The way our synthetic data is produced through FSA's may look contrived, but there are good motivations for using automata in such a study as ours.

Consider the following problem: you are given some RNN r that produces sequences x over a vocabulary V, with probabilities r but you would like to filter out sequences that *do not* contain a specific symbol a, while preserving the relative probabilities of sequences provided by the RNN: $p_{filtered}(x) \propto P_{filtered}(x) = r(x)\ I[a \in x]$. There appears to be no obvious way to realize $p_{filtered}$ through an RNN, apart from techniques similar to what we have been describing in our discussion of Training-2.

The situation is completely different with FSA's. If you have a PFSA (Probabilistic FSA) r_{pfsa} generating sequences x, then you can intersect r_{pfsa} with an automaton that accepts all sequences containing at least one a, and renormalize the intersection through dynamic programming, obtaining a new PFSA that generates the filtered distribution.[8] Such dynamic programming, with the capacity to *anticipate* properties that need to be satisfied on the global sequence, is unavailable in the RNN world.

With RNNs, the situation is reminiscent of RL, with a reward associated with having observed an a during the production of the sequence. But a standard RL approach would mean that we would try to *maximize* $P_{filtered}(x)$, without taking into consideration the original $r(x)$ that we are filtering from. To be correct, we need to find a policy $\pi_\theta(x)$ (similar to an RNN), that tries to *approximate* $p_{filtered}(x)$ in a *distributional* sense, not in a *maximization* sense (see (Bellemare et al., 2017) for related considerations). This is what we try to do in Training-2, using motifs as our main case-study, instead of a single symbol a (which would not make sense for binary strings).

The advantages of using PFSAs in our study are multiple. They provide a well-understood comparison point to the more complex techniques that need to be deployed for autoregressive models. From an operational viewpoint, they also permit, through dynamic programming, to perform various calculations of interest for our study, such as sampling datasets of arbitrary size and computing exact entropy and partition functions that can serve as comparison points for the results obtained with GAMs. In the present paper, we only exploited PFSA's in the context of motifs, but they provide a much larger class of models that could serve to expand our understanding of sequence-based energy based models.

A.2.1 Computing the Entropy of a PFSA

As mentioned earlier, one advantage of using weigthed finite-state automata for generating synthetic data is that some important quantities, such as entropy, mean sequence length, or partition function can be computed by dynamic programming.

Here we only derive a simple iterative method for computing the entropy of a PFSA, the other computations are very similar.[9]

[8] And this is exactly what we do to produce training data for our experiments, but using a binary sequence (motif m) instead of a single symbol a.

[9] For another technique, and for extensions to the computation of relative entropy, see (Carrasco, 1997).

We consider a PFSA with transitions of the form (q, l, q', w), where q, q' are states, l is the label of the transition from q to q' (in our case $l \in \{0, 1\}$), and w is the probability of the transition. The fact that the automaton is *probabilistic*, instead of simply *weighted*, means that the sum of w's associated with transitions starting at q is equal to 1. We further assume that the automaton is *deterministic*, namely that given q and l uniquely determines the next state q'.[10]

The entropy $H(q)$ of a state q is defined as $H(q) \doteq -\sum_{x \succ q} p(x|q) \log p(x|q)$, where $x \succ q$ denotes a sequence of labels x that ends in a final state of the automaton, for which $p(x|q)$ is computed in the obvious way. The entropy of the automaton as a whole is then defined as $H(q_s)$, where q_s is the initial state of the automaton.

Lemma *The entropies of states satisfy the fixpoint equation:*

$$H(q) = \sum_{(q,l,q',w)} -w \log w + wH(q'). \quad (7)$$

Proof. Let's denote by q_l the state obtained from q by following label l. We have:

$$\begin{aligned}
H(q) &= -\sum_{x \succ q} p(x|q) \log p(x|q) \\
&= -\sum_{l \cdot y \succ q} p(l \cdot y|q) \log p(l \cdot y|q) \\
&= -\sum_{l \cdot y \succ q} p(l|q) p(y|q_l) [\log p(l|q) + \log p(y|q_l)] \\
&= -\sum_{l} p(l|q) \log p(l|q) \sum_{y \succ q_l} p(y|q_l) \\
&\quad -\sum_{l} p(l|q) \sum_{y \succ q_l} p(y|q_l) \log p(y|q_l) \\
&= -\sum_{l} p(l|q) \log p(l|q) + \sum_{l} p(l|q) H(q_l) \\
&= \sum_{(q,l,q',w)} -w \log w + wH(q').
\end{aligned}$$

It is possible to show that the state entropies actually correspond to the *least* fixpoint of equation (7), and this allows a simple iterative algorithm for computing the state entropies: at time $t = 0$, for all states q, we define $H^{t=0}(q) \doteq 0$, and then we iterate until convergence:

$$H^{t+1}(q) = \sum_{(q,l,q',w)} -w \log w + wH^{t}(q').$$

[10] The case of non-deterministic probabilistic automata appears much more difficult (Cortes et al., 2008).

A.3 Additional Experiments and Results

(See next pages)

Table 4: Cyclical training vs two stage training for motif $10001011111000, D_m, ft = 1001111$; CE is short for $\text{CE}(T, \pi_\theta)$.

$\lvert D\rvert$	$\frac{\text{CE}_{rs}}{\text{CE}_{rs}\text{(cycl)}}$	$\frac{\text{time}_{rs}}{\text{time}_{rs}\text{(cycl)}}$	$\frac{\text{CE}_{snis}}{\text{CE}_{snis}\text{(cycl)}}$	$\frac{\text{time}_{snis}}{\text{time}_{snis}\text{(cycl)}}$
500	1.02	1.21	1.02	1.51
1000	1.0	1.48	1.08	2.04
5000	1.04	0.57	1.0	0.57
10000	0.98	1.47	1.02	0.45
20000	0.99	2.65	1.0	0.28

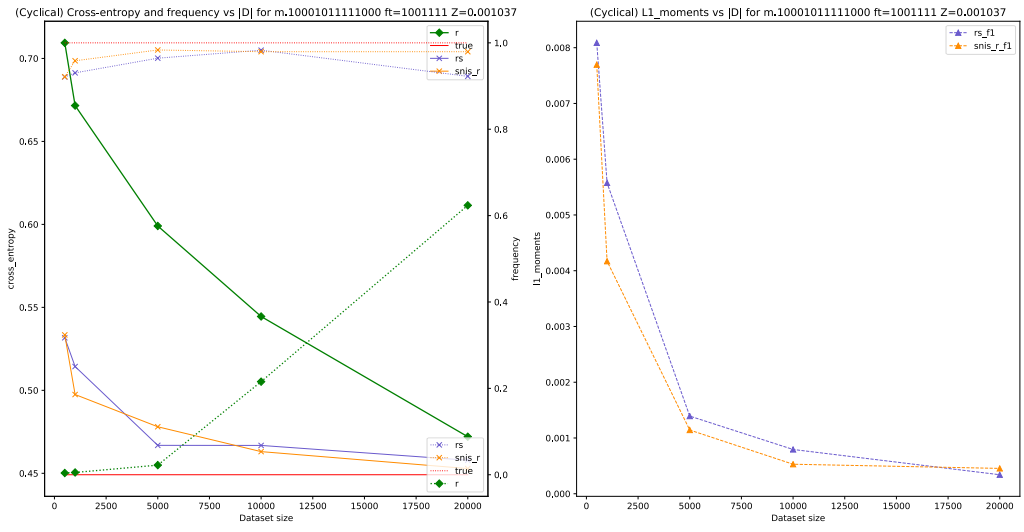

Figure 3: (Cyclical training) Cross-entropy in nats per character, ℓ_1_mom and frequency of sampling motif, depending on the $|D|$, while all distractive features and motif feature are 1: $ft_{[4:7]} = \{1111\}$, $ft_{[1]} = 1$.

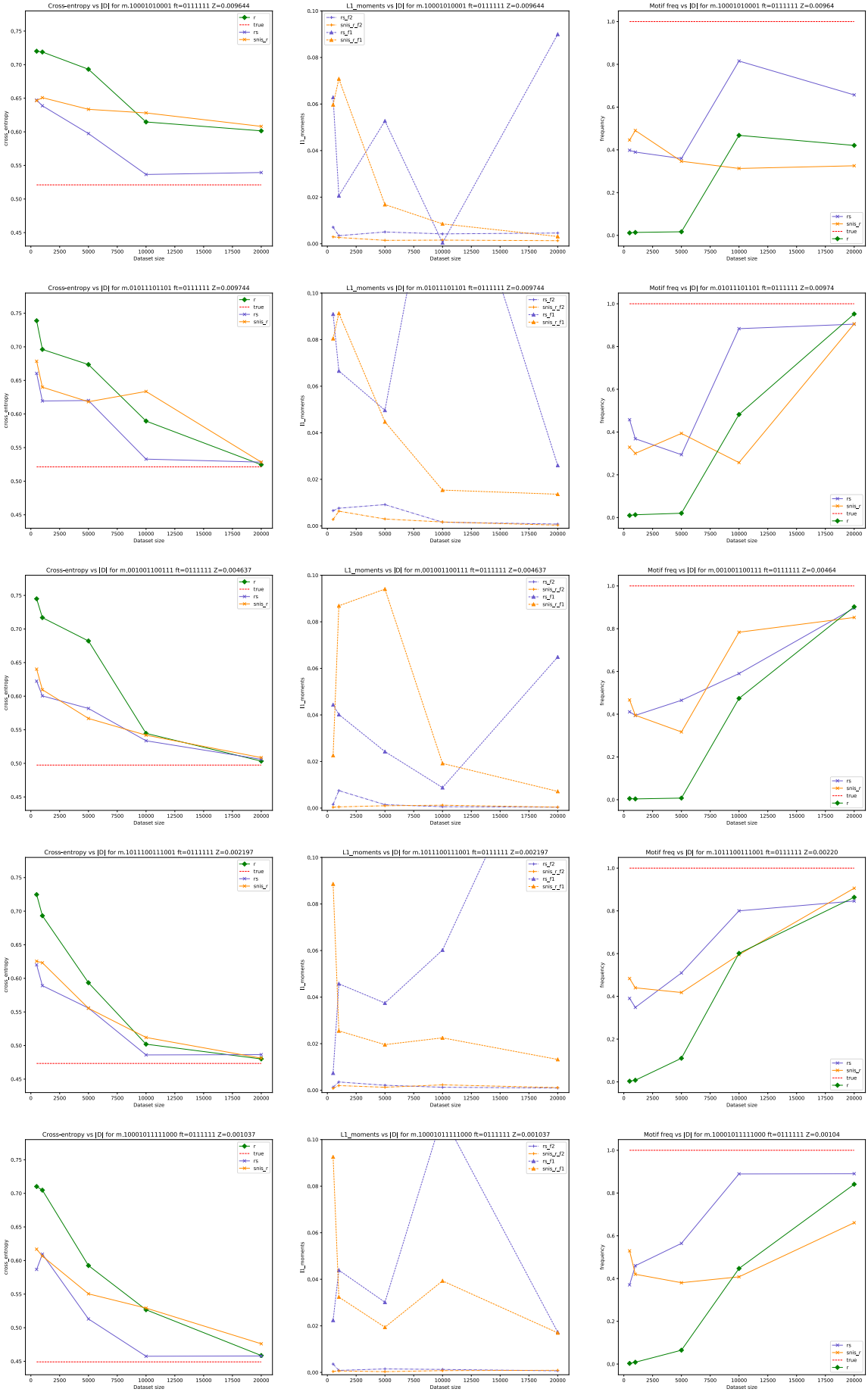

Figure 4: Column 1: Cross-entropy; column 2: ℓ_1_mom; column 3: frequency of sampling motif, depending on the $|D|$, all distractive features are 1: $ft_{[4:7]} = \{1111\}$. Setting: supermotif+submotif, pure D, while varying the rareness of the motif (Z).

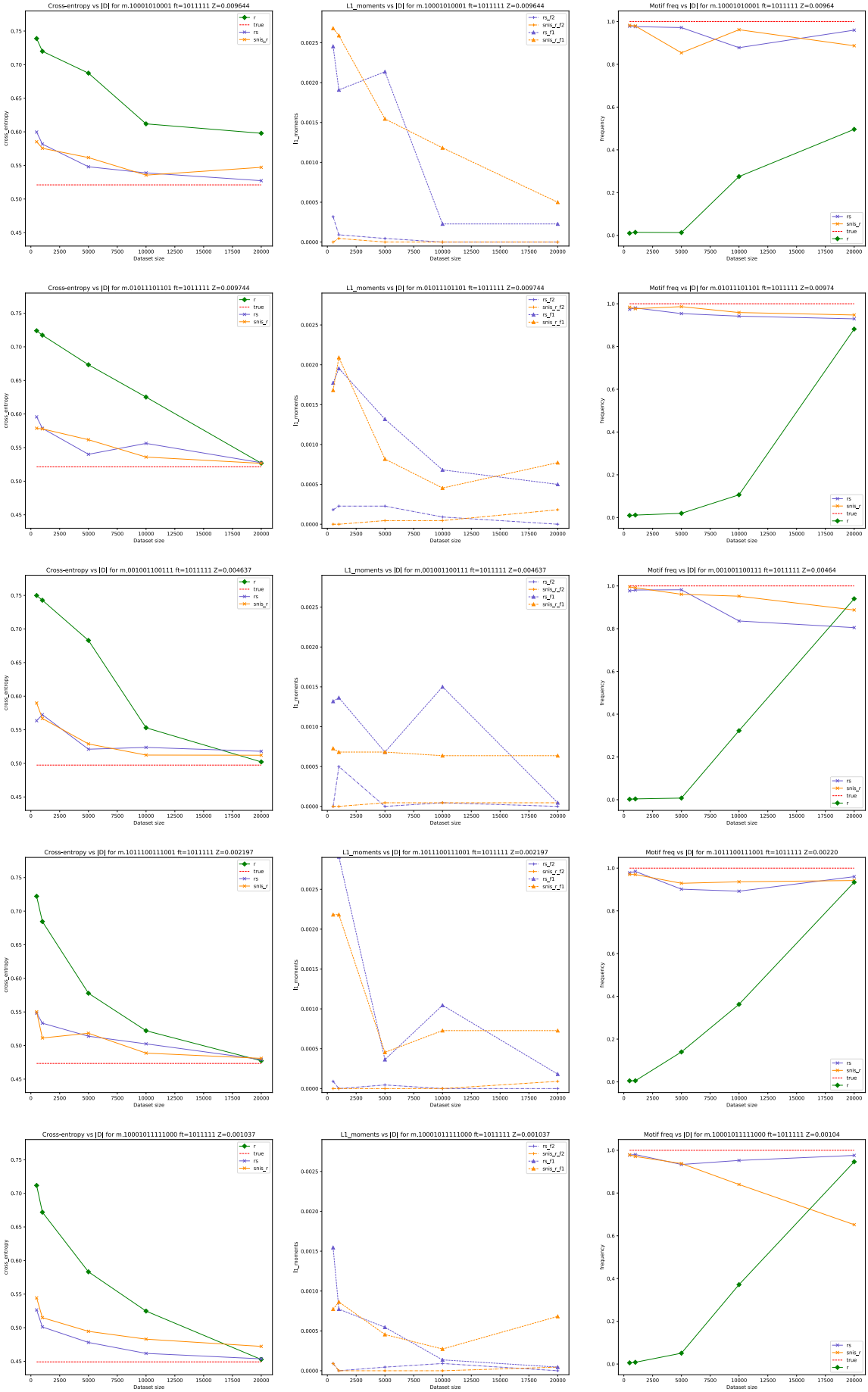

Figure 5: Column 1: Cross-entropy; column 2: ℓ_1_mom; column 3: frequency of sampling motif, depending on the $|D|$, all distractive features are 1: $ft_{[4:7]} = \{1111\}$. Setting: motif+submotif, pure D, while varying the rareness of the motif (Z).

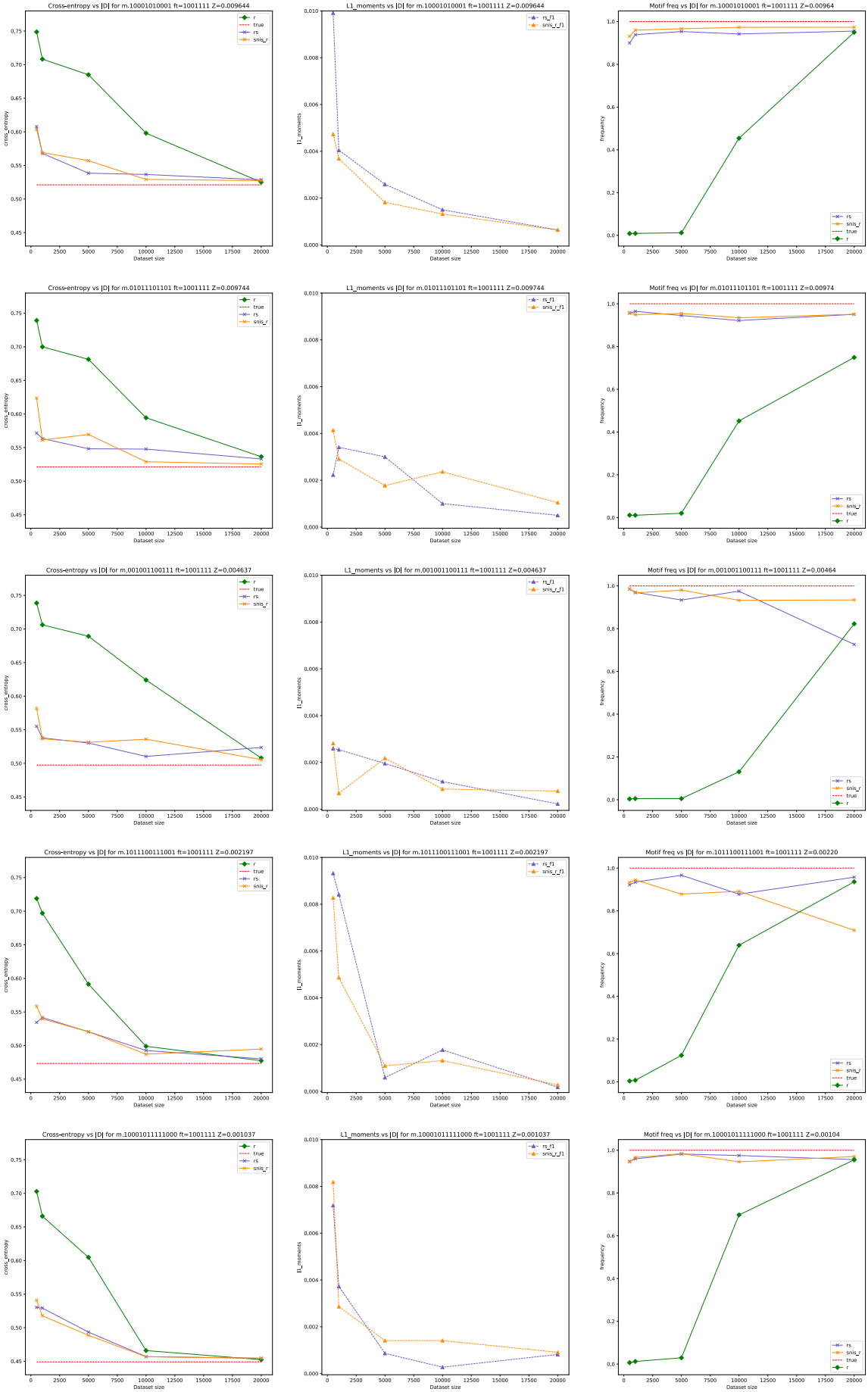

Figure 6: Column 1: Cross-entropy; column 2: ℓ_1_mom; column 3: frequency of sampling motif, depending on the $|D|$, all distractive features are 1: $ft_{[4:7]} = \{1111\}$. Setting: motif, pure D, while varying the rareness of the motif (Z).

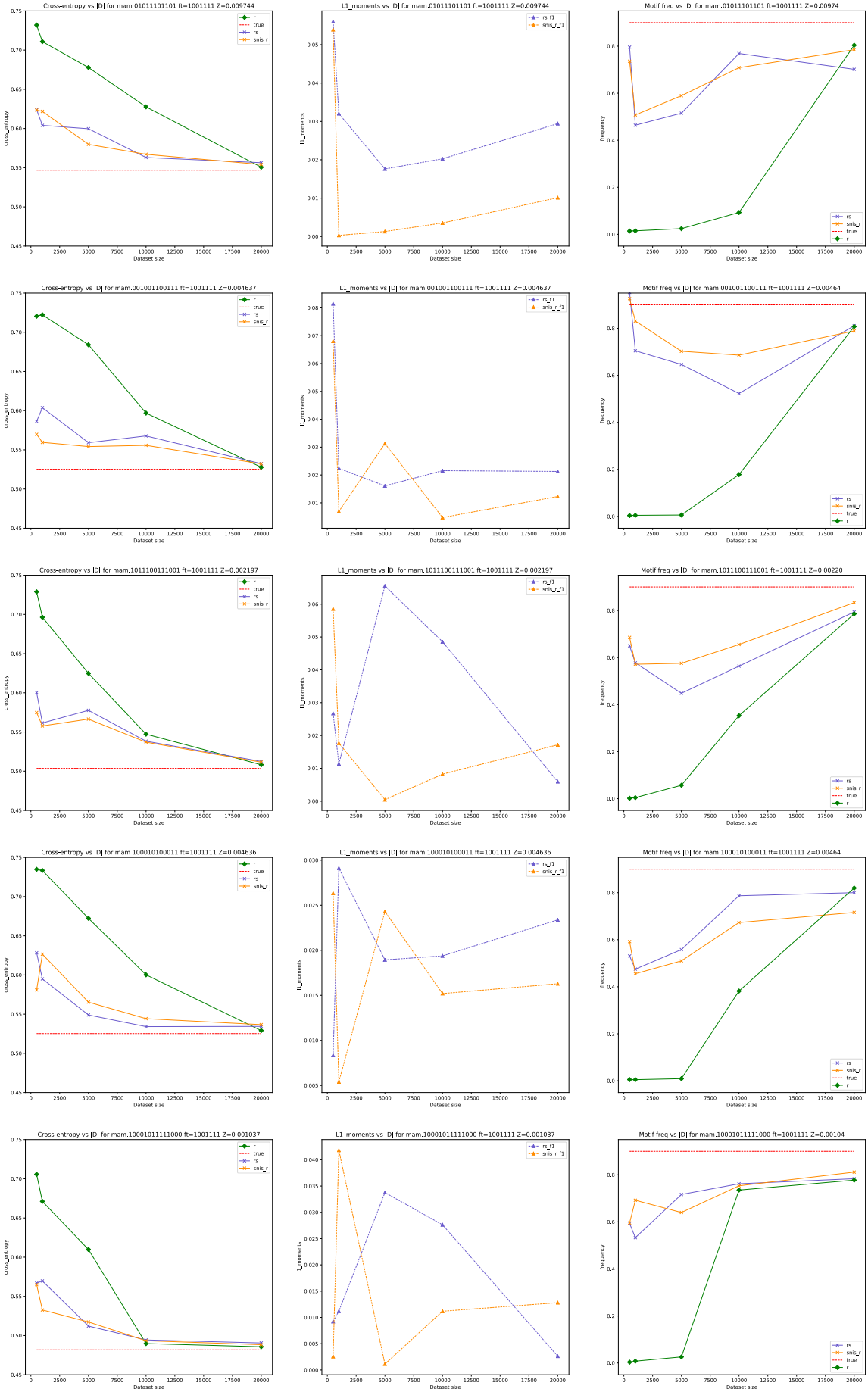

Figure 7: Column 1: Cross-entropy; column 2: ℓ_1_mom; column 3: frequency of sampling motif, depending on the $|D|$, all distractive features are 1: $ft_{[4:7]} = \{1111\}$. Setting: motif, mixture D, while varying the rareness of the motif (Z).