E Applications of Sampling

In this paper, we evaluate our sampling algorithms "intrinsically" by how well a sample approximates the model distribution p_θ—rather than "extrinsically" by using the samples in some larger method.

That said, §1.3 did list some larger methods that make use of sampling. We review them here for the interested reader.

Minimum-risk decoding seeks the output

$$\operatorname*{argmin}_{\mathbf{z}} \sum_{\mathbf{y}} p_\theta(\mathbf{y} \mid \mathbf{x}) \cdot \text{loss}(\mathbf{z} \mid \mathbf{y}) \tag{47}$$

In the special case where $\text{loss}(\mathbf{z} \mid \mathbf{y})$ simply asks whether $\mathbf{z} \neq \mathbf{y}$, this simply returns the "Viterbi" sequence $\mathbf{y}$ that maximises $p_\theta(\mathbf{y} \mid \mathbf{x})$. However, it may give a different answer if the loss function gives partial credit (when $\mathbf{z} \approx \mathbf{y}$), or if the space of outputs $\mathbf{z}$ is simply coarser than the space of taggings $\mathbf{y}$—for example, if there are many action sequences $\mathbf{y}$ that could build the same output structure $\mathbf{z}$. In these cases, the optimal $\mathbf{z}$ may win due to the combined support of many suboptimal $\mathbf{y}$ values, and so finding the optimal $\mathbf{y}$ (the Viterbi sequence) is not enough to determine the optimal $\mathbf{z}$.

The risk objective (47) is a expensive expectation under the distribution $p_\theta(\mathbf{y} \mid \mathbf{x})$. To approximate it, one can replace $p_\theta(\mathbf{y} \mid \mathbf{x})$ with an approximation $\hat{p}(\mathbf{y})$ that has small support so that the summation is efficient. Particle smoothing returns such a $\hat{p}$—a non-uniform distribution (28) over M particles. Since those particles are randomly drawn, $\hat{p}$ is itself stochastic, but $\mathbb{E}\left[\hat{p}(\mathbf{y})\right] \approx p_\theta(\mathbf{y} \mid \mathbf{x})$, with the approximation improving with the quality of the proposal distribution (which is the focus of this paper) and with M.

In *supervised* training of the model (1) by maximizing conditional log-likelihood, the gradient of $\log p(\mathbf{y}^* \mid \mathbf{x})$ on a single training example $(\mathbf{x}, \mathbf{y}^*)$ is $\nabla_\theta \log p_\theta(\mathbf{y}^* \mid \mathbf{x}) = \nabla_\theta G_T^* - \sum_{\mathbf{y}} p_\theta(\mathbf{y} \mid \mathbf{x}) \cdot \nabla_\theta G_T$. The sum is again an expectation that can be estimated by using $\hat{p}$. Since $\mathbb{E}\left[\hat{p}(\mathbf{y})\right] \approx p_\theta(\mathbf{y} \mid \mathbf{x})$, this yields a stochastic estimate of the gradient that can be used in the stochastic gradient ascent algorithm (Robbins and Monro, 1951).[21]

In *unsupervised or semi-supervised training* of a generative model $p_\theta(\mathbf{x}, \mathbf{y})$, one has some training examples where $\mathbf{y}^*$ is unobserved or observed incompletely (e.g., perhaps only $\mathbf{z}$ is observed). The Monte Carlo EM algorithm for estimating θ (Wei and Tanner, 1990) replaces the missing $\mathbf{y}^*$ with samples from $p_\theta(\mathbf{y} \mid \mathbf{x}, \text{partial observation})$ (this is the Monte Carlo "E step"). This *multiple imputation* procedure has other uses as well in statistical analysis with missing data (Little and Rubin, 1987).

Modular architectures provide another use for sampling. If $p_\theta(\mathbf{y} \mid \mathbf{x})$ is just one stage in an NLP annotation *pipeline*, Finkel et al. (2006) recommend passing a diverse sample of $\mathbf{y}$ values on to the next stage, where they can be further annotated and rescored or rejected. More generally, in a *graphical model* that relates multiple strings (Bouchard-Côté et al., 2007; Dreyer and Eisner, 2009; Cotterell et al., 2017), inference could be performed by particle belief propagation (Ihler and McAllester, 2009; Lienart et al., 2015), or with the help of stochastic-inverse proposal distributions (Stuhlmüller et al., 2013). These methods call conditional sampling as a subroutine.

[21] Notice that the gradient takes this "difficult" form only because the model is globally normalized. If we were training a locally normalized conditional model (McCallum et al., 2000), or a locally normalized joint model like equation (4), then sampling methods would not be needed, because the gradient of the (conditional or joint) log-likelihood would decompose into T "easy" summands that each involve an expectation over the small set of y_t values for some t, rather than over the exponentially larger set of strings $\mathbf{y}$. However, this simplification goes away outside the fully supervised case, as the next paragraph discusses.

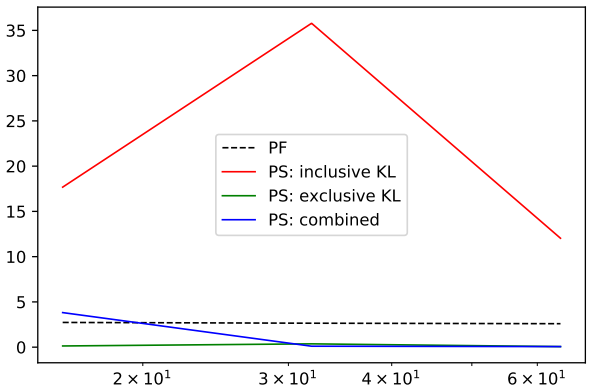

Figure 4: Offset KL divergence on the *last char* task: a pathological case where a naive particle filtering sampler does really horribly, and an ill-trained smoothing sampler even worse. The logarithmic x-axis is the particle size used to train the sampler. At test time we evaluate with the same particle size ($M = 32$).

F Effect of different objective functions on lookahead optimization

§5 discussed inclusive and exclusive KL divergences, and gave our rationale for optimizing an interpolation of the two. Here we study the effect of the interpolation weight. We train the lookahead sampler, and the joint language model, on a toy problem called "last char," where $\mathbf{y}$ is a deterministic function of $\mathbf{x}$: either a lowercased version of $\mathbf{x}$, or an identical copy of $\mathbf{x}$, depending on whether the last character of $\mathbf{x}$ is `0` or `1`. Note that this problem requires lookahead.

We obtain our $\mathbf{x}$ sequences by taking the phoneme sequence data from the stressed syllable tagging task and flipping a fair coin to decide whether to append `0` or `1` to each sequence. Thus, the dataset may include $(\mathbf{x}, \mathbf{y})$ pairs such as (`K AU CH 0`, `k au ch 1`) or (`K AU CH 1`, `K AU CH 1`), but not (`K AU CH 1`, `k au ch 1`).

We treat this as a tagging problem, and treat it with our tagging model in §6.1. Results are in Figure 4. We see that optimizing for $\mathrm{KL}(\hat{p}||q)$ at a low particle size gives much worse performance than other methods. On the other hand, the objective function $\mathrm{KL}(q||p)$ achieves constantly good performance. The middle ground $\frac{\mathrm{KL}(\hat{p}||q)+\mathrm{KL}(q||p)}{2}$ improves when the particle size increases, and achieves better results than $\mathrm{KL}(q||p)$ at larger particle sizes.

G Generative process for source separation

Given an alphabet Σ, J strings $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(J)} \in \Sigma^*$ are independently sampled from the respective distributions $p^{(1)}, \ldots p^{(J)}$ over Σ^* (possibly all the same distribution $p^{(1)} = \cdots = p^{(J)}$). These source strings are then combined into a single observed string $\mathbf{x}$, of length $K = \sum_j K_j$, according to an **interleaving string** $\mathbf{y}$, also of length K. For example, $\mathbf{y} = 1132123$ means to take two characters from $\mathbf{x}^{(1)}$, then a character from $\mathbf{x}^{(3)}$, then a character from $\mathbf{x}^{(2)}$, etc. Formally speaking, $\mathbf{y}$ is an element of the mix language $\mathcal{Y}_{\mathbf{x}} = \textsc{mix}(1^{k_1}, 2^{k_2}, \ldots, j^{k_j})$, and we construct $\mathbf{x}$ by specifying the character $x_k \in \Sigma$ to be $x^{(y_k)}_{|\{i \le k : y_i = y_k\}|}$. We assume that $\mathbf{y}$ is drawn from some distribution over $\mathcal{Y}_{\mathbf{x}}$. The source separation problem is to recover the interleaving string $\mathbf{y}$ from the interleaved string $\mathbf{x}$.

We assume that each source model $p^{(j)}(\mathbf{x}^{(j)})$ is an RNN language model—that is, a locally normalized state machine that successively generates each character of $\mathbf{x}^{(j)}$ given its left context. Thus, each source model is in some state $\mathbf{s}^{(j)}_t$ after generating the prefix $\mathbf{x}^{(j)}_{:t}$. In the remainder of this paragraph, we suppress the superscript $^{(j)}$ for simplicity. The model now stochastically generates character x_{t+1} with probability $p(x_{t+1} \mid \mathbf{s}_t)$, and from $\mathbf{s}_t$ and this x_{t+1} it deterministically computes its new state $\mathbf{s}_{t+1}$. If x_{t+1} is a special "end-of-sequence" character EOS, we return $\mathbf{x} = \mathbf{x}_{:t}$.

Given only $\mathbf{x}$ of length T, we see that $\mathbf{y}$ could be any element of $\{1, 2, \ldots, J\}^T$. We can write the posterior probability of a given $\mathbf{y}$ (by Bayes' Theorem) as

$$p(\mathbf{y} \mid \mathbf{x}) \propto p(\mathbf{y}) \prod_{j=1}^{J} p^{(j)}\left(\mathbf{x}^{(j)}\right) \quad (48)$$

where (for this given $\mathbf{y}$) $\mathbf{x}^{(j)}$ denotes the subsequence of $\mathbf{x}$ at indices k such that $y_k = j$. In our experiments, we assume that y was drawn uniformly from $\mathcal{Y}_{\mathbf{x}}$, so $p(\mathbf{y})$ is constant and can be ignored. In general, the set of possible interleavings $\mathcal{Y}_{\mathbf{x}}$ is so large that computing the constant of proportionality (partition function) for a given $\mathbf{x}$ becomes prohibitive.