E Applications of Sampling

In this paper, we evaluate our sampling algorithms “intrinsically” by how well a sample approximates the model distribution \( p_\theta \)—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

\[
\arg\min_z \sum_y p_\theta(y \mid x) \cdot \text{loss}(z \mid y)
\]  

(47)

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

The risk objective (47) is an expensive expectation under the distribution \( p_\theta(y \mid x) \). To approximate it, one can replace \( p_\theta(y \mid x) \) with an approximation \( \hat{p}(y) \) 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 \( \mathbb{E}[\hat{p}(y)] \approx p_\theta(y \mid x) \), 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 \( \log p(y^* \mid x) \) on a single training example \((x, y^*)\) is \( \nabla_\theta \log p_\theta(y^* \mid x) = \nabla_\theta G_T - \sum_y p_\theta(y \mid x) \cdot \nabla_\theta G_T \). The sum is again an expectation that can be estimated by using \( \hat{p} \). Since \( \mathbb{E}[\hat{p}(y)] \approx p_\theta(y \mid x) \), this yields a stochastic estimate of the gradient that can be used in the stochastic gradient ascent algorithm (Robbins and Monro, 1951).\(^{21}\)

---

\(^{21}\)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 \( y_i \) values for some \( i \), rather than over the expon-

tially larger set of strings \( y \). However, this simplification goes away outside the fully supervised case, as the next paragraph discusses.
§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 $y$ is a deterministic function of $x$: either a lowercased version of $x$, or an identical copy of $x$. We obtain our results by taking the sampled from the respective distributions $p(1), \ldots, p(J)$ over $\Sigma^*$ (possibly all the same distribution $p(1) = \cdots = p(J)$). These source strings are then combined into a single observed string $x$, of length $K = \sum_j K_j$, according to an interleaving string $y$, also of length $K$. For example, $y = 1132123$ means to take two characters from $x^{(1)}$, then a character from $x^{(3)}$, then a character from $x^{(2)}$, etc. Formally speaking, $y$ is an element of the mix language $Y_x = \text{MIX}(1^{k_1}, 2^{k_2}, \ldots, J^{k_J})$, and we construct $x$ by specifying the character $x_k \in \Sigma$ to be $x^{(y_k)}_{[i \leq k_0 = y_k]}$. We assume that $y$ is drawn from some distribution over $Y_x$. The source separation problem is to recover the interleaving string $y$ from the interleaved string $x$.

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

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

$$p(y \mid x) \propto p(y) \prod_{j=1}^{J} p^{(j)}(x^{(j)})$$

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