Particle-filter based language models

December 13, 2017

Rachit Singh, Yuntian Deng
rachitsingh@college.harvard.edu

There's significant interest in finding generative models of text using latent variable models, since latent variables often let us control generation by making small steps in the latent space. One of the simplest problems to solve in this way is language modeling, the problem of computing a probability distribution $p(x_1, \dots, x_T)$ over sentences $\mathbf{x} = \{x_1, \dots, x_T\}$. The state of the art in language modeling is RNN-based language models (RNN-LMs), which do not incorporate explicit inference over latent variables.

Instead, we build on the theory of state-space models and recent work on generative models to build a latent variable language model for text, which we train using variational inference. Explicitly, we look at the case of generative model with *sequential latent variables*, or a sequence $\{z_1, \dots, z_T\}$ of latent variables corresponding to a sentence $\{x_1, \dots, x_T\}$. We give more details on our explicit assumptions below in Section 2.

1 Prior work

There are two branches of prior work that we consider. First, the state of the art (SOTA) in language modeling are RNN-LMs, where the probability distribution factors over each word conditional on the past, and we compute the conditional probabilities using an RNN. The main techniques used to achieve SOTA are:

- **Regularization methods to prevent overfitting**: Merity et al. (2017) explores a set of regularization methods, including DropConnect (dropout inside the RNN cell), variational dropout (sampling the dropout once per RNN rollout), embedded dropout, and weight tying.
- **Improved optimization methods** Merity et al. (2017) mentions BPTT jitter (randomizing the BPTT window) and averaged SGD (ASGD) as techniques that improve performance.
- **Removing softmax limitations** Recently, Yang et al. (2017) explores the the restriction that using a softmax to generate probabilities for a language model. Essentially, they show that the probability distributions representable by a softmax is restricted to rank-d matrices, where d is the embedding size [1]. They overcome this limitation using a mixture-of-softmaxes, while retaining roughly the same number of parameters.

[1]They show that if the true probability distribution over a sequence of contexts (e.g., a sentence) is representated by a matrix $\mathbf{A}$, then the distribution can be represented only if there's a factorization into two matrices $\mathbf{H}_\theta \mathbf{W}_\theta^T$ that is off from $\mathbf{A}$ only by a per-row shift (i.e. the bias).

This is excellent experimental work that helps bring language model training in line with the kinds of methods that have made image tasks tractable, like batch normalization [2]. The underlying generative model is essentially still the same as that of (cite: Mikolov), i.e. an RNN-LM. We intend to use these methods to bring our model to state of the art.

The other branch of prior work is work on sequential generative models, which assume a particular generative process and approximate the posterior using variational inference (VI). There are many attempts in the literature to introduce sequential latent variables in the generative process. Bowman et al. (2015); Zhang et al. (2016); Li et al. (2017) studied applying VI to generative models over text, and use the hidden state after encoding as the latent variable. More recently, (Bayer and Osendorfer, 2014; Gregor et al., 2015; Chung et al., 2015; Fraccaro et al., 2016). As a whole, these works find two problems with applying these models to text: (1) at the beginning of training, the KL divergence loss is too strong, causing the model to make the variational approx. q to converge to the prior p, and become an RNN-LM (Bowman et al., 2015), and (2) Yang et al. (2017) makes the claim that for discrete data "the variational lower bound is usually too loose to yield a competitive approximation compared to standard auto-regressive models". (1) is usually solved by annealing the KL portion of the loss from 0 to 1, essentially removing noise early in training. For (2), Maddison et al. (2017); Naesseth et al. (2017); Le et al. (2017) examine the issue of the looseness of the variational lower bound in sequential models, and show that the issue is that errors made in the inference process are propagated forwards without correction, making it unlikely that the full sampled sequence $\{z_1, \ldots, z_T\}$ has high posterior probability. They develop a particle-filter based variational inference, which has significant improvement over variational RNNs trained using the ELBO or IWAE as objectives. There is a more detailed overview in Section 4.

A high level overview of our proposal is to build a language model for text that follows the RNN with sequential latent state paradigm, and fit it using variational inference. We'll use the solutions above, which are (1) anneal KL to encourage use of the latent states, and (2) use the particle-filter method to optimize a tighter lower bound than IWAE or ELBO. Given the wealth of recent knowledge about training RNN-LMs, we'll experiment with those regularization techniques to see if they lead to improved performance. One recent work (Zheng et al., 2017) proposes a very similar method which does not outperform an LSTM baseline. However, their work does not condition the distribution of the word x_t on past generations, which is well known to be important to language modeling.

2 Model

First, for a particular sentence of length T, we have a sequence of tokens $\{x_1, \ldots, x_T\}$. We assume that there is a sequence of latent variables $\{z_1, \ldots, z_T\}$, and that the joint distribution factors as:

$$p(x_1, \ldots, x_T, z_1, \ldots, z_T) = \prod_{t=1}^{T} p(z_t | z_{<t}) p(x_t | z_t, z_{<t}, x_{<t})$$

[2]There's work (Cooijmans et al., 2016) that extends batch normalization to the recurrent setting, but I had trouble getting it to work on NMT tasks.

$$\begin{tikzcd}
\mathbf{s}_0 \arrow[r] & \mathbf{s}_1 \arrow[r] \arrow[d, squiggly] & \mathbf{s}_2 \arrow[r] \arrow[d, squiggly] & \mathbf{s}_3 \arrow[r] \arrow[d, squiggly] & \cdots \\
 & \mathbf{z}_1 \arrow[ur] \arrow[d] & \mathbf{z}_2 \arrow[ur] \arrow[d] & \mathbf{z}_3 \arrow[ur] \arrow[d] & \cdots \\
\mathbf{r}_0 \arrow[r] & \mathbf{r}_1 \arrow[r] \arrow[d, squiggly] & \mathbf{r}_2 \arrow[r] \arrow[d, squiggly] & \mathbf{r}_3 \arrow[r] \arrow[d, squiggly] & \cdots \\
 & \mathbf{x}_1 \arrow[ur] & \mathbf{x}_2 \arrow[ur] & \mathbf{x}_3 \arrow[ur] & \cdots
\end{tikzcd}$$

Figure 1: The proposed generative process. Straight lines indicate deterministic functions, while squiggly arrows indicate parametrizing sampled variables, which are circled.

This relaxes the assumptions of state-space and state-space LSTM models by allowing a more complicated transition model and generation model. Specifically, we allow the distribution for the current word x_t to depend on previously generated words (and so, generalize an RNN-LM), and also allow the distribution over current state to depend on previous states. So, we can compute the probabilities $p(z_t|z_{<t})$ and $p(x_t|z_t, z_{<t}, x_{<t})$ using RNNs as follows:

Generation

First, initialize s_0, r_0 from a trainable variable. Then, for each time step $t \in \{1, \dots, T\}$:

1. Perform an RNN transition $s_t = \mathrm{RNN}_s(s_{t-1}, z_t), r_t = \mathrm{RNN}_r(r_{t-1}, x_{t-1}, z_t)$.
2. We sample $z_t \sim \mathcal{N}(s_t, I)$, $x_t \sim \mathrm{Cat}(r_t^\top \theta)$. In other words, we let $p(z_t|z_{<t}) = \mathcal{N}(z_t; s_t, I)$ and $p(x_t|z_t, z_{<t}, x_{<t}) = \mathrm{Cat}(x_t; r_t^\top \theta)$

Note that the internal variables of RNN_s actually encode the prior distribution $p(z_1, \dots, z_T)$ of the latent states. We use spherical normal distributions here to simplify the model description. It's possible that we might want to use s_t, r_t to parametrize the mean and the variances together, but this takes very little modification. The input to RNN_r is the previous hidden state, and also x_{t-1}, z_t, which we concatenate as appropriate. The distribution for x_t, a categorical, is over the vocabulary; θ refers to the embedding parameters, so the categorical distribution's entries are determined via a softmax as

$$\begin{tikzcd}
 & \mathbf{x}_1 \arrow[d] & \mathbf{x}_2 \arrow[d] & \mathbf{x}_3 \arrow[d] & \cdots \\
 & \mathbf{b}_1 \arrow[d] \arrow[r, shift right] & \mathbf{b}_2 \arrow[l, shift right] \arrow[d] \arrow[r, shift right] & \mathbf{b}_3 \arrow[l, shift right] \arrow[d] \arrow[r, shift right] & \cdots \arrow[l, shift right] \\
\mathbf{h}_0 \arrow[r] & \mathbf{h}_1 \arrow[r] \arrow[d, squiggly] & \mathbf{h}_2 \arrow[r] \arrow[d, squiggly] & \mathbf{h}_3 \arrow[r] \arrow[d, squiggly] & \cdots \\
 & \mathbf{z}_1 \arrow[ur] & \mathbf{z}_2 \arrow[ur] & \mathbf{z}_3 \arrow[ur] & \cdots
\end{tikzcd}$$

Figure 2: The proposed inference network. The same convention from above is used: straight lines indicate deterministic functions, squiggly arrows indicate parametrization for sampled variables. The two-way arrows for the $\mathbf{b}_i$ indicates that they're hidden states from a bidirectional RNN.

3 Inference

For inference, we use a variational network. We assume that our approximate posterior distribution $q(z_1, \ldots, z_T) \approx p(z_1, \ldots z_T | x_1, \ldots, x_T)$ factors as follows:

$$q(z_1, \ldots, z_T) = \prod_{t=1}^{T} q(z_t | z_{<t})$$

Then, we can compute the probability distribution $q(z_t | z_{<t})$ as follows:

Inference

1. Run a bidirectional RNN over the words $x_1, \ldots, x_T$, and let their corresponding hidden states be $b_1, \ldots, b_T$
2. Let h_0 be a trainable vector. For each time step $t \in \{1, \ldots, T\}$:
   (a) Perform an RNN transition: $h_t = \mathrm{RNN}_h(h_{t-1}, b_t)$
   (b) Sample a new $z_t \sim \mathcal{N}(h_t, I)$. In other words, $q(z_t | z_{<t}) = \mathcal{N}(z_t; h_t, I)$

One unfortunate side effect of using a particle filter for tightening the lower bound is that we must sample forwards, so we sample z_1 before z_2, etc. However, it's often beneficial to do inference in reverse, as explained by Sønderby et al. (2016) in *ladder variational autoencoders* (LVAE) and Krishnan et al. (2017). Maddison et al. (2017) discuss this using the term 'sharpness' and show that this effect doesn't change the fact that the particle filter is better

- even if we include 'smoothing' terms for a regular ELBO or IWAE training, it still performs worse.

While we have a somewhat clearer sense of what the generative model should look like, we have a lot of choice over *what* our hidden variables are. For example, we can assume that they they're Gaussian, or possible a discrete random variable.

We also have significant choice over the prior in these case. Since we want the prior to be over the entire vector $\mathbf{z}$, we can factor the prior over time as well.

Preliminary optimization

As a first step, we can optimize either (1) the regular ELBO under a regular sampling procedure, and (2) the normal IWAE across time steps. Here we'll give a description of (1), the other case is similar:

ELBO optimization

1. Sample a set of variables $\{z_1, \ldots, z_T\}$ as described in the "Inference".

2. Compute the negative ELBO:

$$\mathcal{L}_{\text{ELBO}} = -\mathbb{E}_{\mathbf{z}}\left[\sum_{t=1}^{T} \underbrace{\log p(x_t|z_t, z_{<t}, x_{<t})}_{\text{RNN}_r} + \underbrace{\log p(z_t|z_{<t})}_{\text{RNN}_s} - \underbrace{\log q(z_t^i|z_{<t})}_{\text{RNN}_h}\right]$$

3. Compute the gradient of the ELBO, via reparametrization gradients for q, and perform an update

---

One thing to note is that we will almost definitely need to anneal the KL as in Bowman et al. (2015). This corresponds to the second and third terms. In the IWAE case, we'll just compute this quantity over K samples, and take the logsumexp of the values and add $\log K$.

4 Particle Filter

Recent work (Naesseth et al. (2017); Maddison et al. (2017); Le et al. (2017)) has shown how to find a significantly tigher variational bound in the case of sequential latent distributions (see Fig. 4). Essentially, by treating the sequence of samples (i.e. collection of samples for z_1, another as z_2, etc.) as the progressive states of a particle filter, we can do better than the naive approach of sampling k separate chains, by using a *resample/reweight* step to remove low weight sequences and resample high weight sequences. In effect, this is a form of probabilistic, differentiable beam search - we can essentially optimize the proposal distribution $p(z_t|\mathbf{z}_{1:t-1}, \mathbf{y})$, but only if the corresponding predictions are good.

Formally, the algorithm to compute the marginal likelihood estimate is found in Section 4.

```
for $t \in \{1, \ldots, T\}$ do
    if t = 1 then
        For each $i \leq K$, sample $z_1^i \sim q(z_1)$ (i.e., using $h_1$).
    else
        if <resample this step>[3] then
            Resample $a_t^i \sim \mathrm{Cat}(w_{t-1}^j / \sum w_{t-1})$
        else
            Let $a_t^i = i$
        end if
        for $1 \leq i \leq K$ do
            Sample $z_t^i \sim q(z_t | z_{<t}^{a_t^i})$
            Define $z_{<t+1} := (z_t^i ; z_{<t}^{a_t^i})$
            Compute weights $w_t^i = w_{t-1}^i \cdot p(x_t | z_t^i, z_{<t}^{a_t^i}, x_{<t}) p(z_t^i | z_{<t}^{a_t^i}) / q(z_t^i | z_{<t}^{a_t^i})$
        end for
        Renormalize $w_t^i \leftarrow w_t^i / \sum_i w_t^i$
    end if
end for
return $\hat{p}(\mathbf{x}) \triangleq \prod_{t=1}^T \frac{1}{K} \sum_{i=1}^K w_t^i$
```

The algorithm returns $\hat{p}(\mathbf{x})$, an unbiased estimate of $p(\mathbf{x})$:

$$\mathbb{E}[\hat{p}(\mathbf{x})] = \mathbb{E}\left[\prod_{t=1}^T \frac{1}{K} \sum_{i=1}^K w_t^i\right] = p(\mathbf{x})$$

We can use this estimate to produce the surrogate ELBO (which is still a lower bound) for optimization:

$$\tilde{\mathcal{L}} \triangleq \mathbb{E}_{\mathbf{z}^{1:K}, \mathbf{a}^{1:K}}\left[\sum_{t=1}^T \log\left(\frac{1}{K} \sum_{i=1}^K w_t^i\right)\right]$$

In Naesseth et al. (2017), this expectation is denoted as over the distribution $\phi(\mathbf{z}^{1:K}, \mathbf{a}^{1:K})$, which is the distribution over sampled $\mathbf{z}$ particles and the ancestor variables $\mathbf{a}$. This factors as:

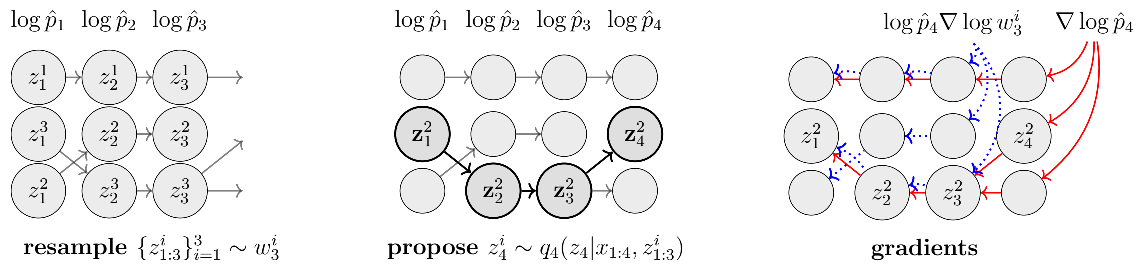
$$\phi(\mathbf{z}^{1:K}, \mathbf{a}^{1:K}) = \underbrace{\left(\prod_{i=1}^K \prod_{t=2}^T \frac{w_{t-1}^{a_t^i}}{\sum_l w_{t-1}^l}\right)}_{\text{ancestor choices}} \underbrace{\left(\prod_{i=1}^K q(z_1^i) \prod_{t=2}^T q(z_t^i | z_{<t}^{a_t^i})\right)}_{\text{posterior samples}}$$

4.0.1 Training

One of the main contributions of the recent work is a method for computing gradients, so we cover that as well. Note that there are 3 RNNs in our model as stated:

1. RNN_s which parametrizes the *prior distribution* over $\mathbf{z}$

2. RNN_r, which parametrizes the *likelihood* of $\mathbf{x}$



$\log \hat{p}_1$ $\log \hat{p}_2$ $\log \hat{p}_3$
resample $\{z^i_{1:3}\}^3_{i=1} \sim w^i_3$
$\log \hat{p}_1$ $\log \hat{p}_2$ $\log \hat{p}_3$ $\log \hat{p}_4$
propose $z^i_4 \sim q_4(z_4|x_{1:4}, z^i_{1:3})$
$\log \hat{p}_4 \nabla \log w^i_3$ $\nabla \log \hat{p}_4$
gradients

Figure 3: Visualizing FIVO (from (Maddison et al., 2017)); (Left) Resample from particle trajectories to determine inheritance in next step, (middle) propose with q_t and accumulate loss $\log \hat{p}_t$, (right) gradients (in the reparameterized case) flow through the lattice, objective gradients in solid red and resampling gradients in dotted blue.

3. RNN$_h$, which parametrizes the *variational posterior* over $\mathbf{z}$

So we need to find the gradient with respect to the parameters of each of these RNNs. We can expand the gradient of the expectation as the sum of two terms:

$$\nabla \tilde{\mathcal{L}} = \mathbb{E}\left[\nabla \log \hat{p}(\mathbf{x}) + \log \hat{p}(\mathbf{x}) \nabla \log \phi(\mathbf{z}^{1:K}, \mathbf{a}^{1:K})\right]$$

The first term is straightforward, since the computation of w^i_t is essentially a ratio of probabilities calculated from the RNNs. However, the second term is a little tricker, since the ancestor variables are discrete, and so their score function gradient is high variance. However, Maddison et al. (2017) notes that dropping this term of the gradient seems to improve training, since variance contributes more noise than signal. In conversation Chris Maddison also confirmed that it wasn't useful to compute those gradients. Our first attempt will avoid computing those gradients to ease implementation, but we'll experiment with those terms later as well.

If all we need to do is use $\mathbb{E}[\nabla \log \hat{p}(\mathbf{x})]$ as our gradient estimator, then there is very little bookkeeping to do in PyTorch. We just need to make sure that the `Variable`s corresponding to the weights retain links to the past when sampled - this works with `.clone()`, or indexing, which is probably the easiest method. See Fig. 3 for a visualization of the calculation of gradients.

Evaluation

There are several common metrics by which we can evaluate the performance, and in order of computational requirements, (and backwards in terms of reliability) they are: (1) use $\tilde{\mathcal{L}}$, (2) use the IWAE estimate, (3) generate from the prior and evaluate the log-likelihood of a holdout set.

5 Discussion

Disadvantages

One primary disadvantage when compared to RNN-LMs or to the seq2seq model of Bowman et al. (2015) is that the addition of another sequential model (i.e. a decoder) for predicting

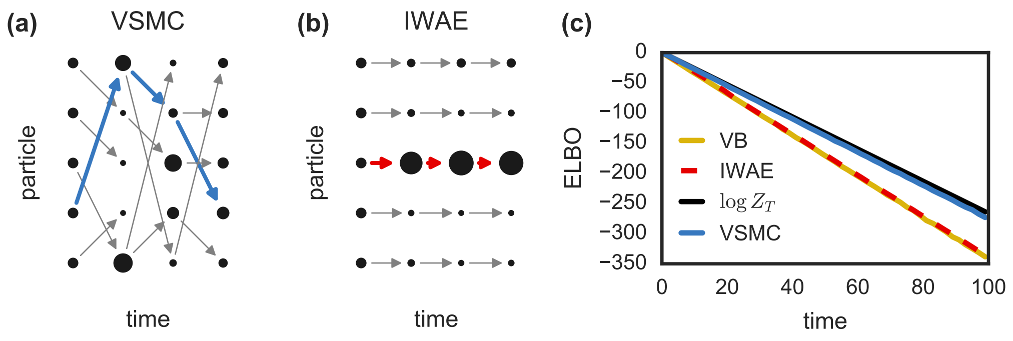


Figure 4: Comparing VSMC and the IWAE (from (Naesseth et al., 2017)) (a) VSMC resamples the particles at each step, and chooses the most likely sequence, akin to beam search. (b) IWAE does the same, but without resampling, and only chooses one of the k chains. This means that only one sequence might be sampled in general. (c) As time increases, IWAE and regular variational Bayes (VB) diverge from the true marginal likelihood, while VSMC stays close.

the latent state distribution increases the number of parameters significantly. Our approach for dealing with this concern is to reduce the size of the latent representation $\mathbf{z}$. Since we're stretching the latent representation over the entire sequence, we can use smaller Gaussian vectors for each particular state. In general using a discrete latent space is difficult for inference, but recent advancements have made this problem less difficult. Still, since we can only sample a very small subset of the possible sequences (and they are totally disjoint), we might have trouble early in the training process.

Advantages

1. **Sequential latent states add significant flexibility to the model**: this is a heuristic argument, but if we believe that there's still some inherent randomness during generation after seeing some portion of the sentence, then we should use sequential latent states. An isotropic Gaussian generative process can't capture dependencies between latent variables, so this model is inherently more flexible.

2. **A tighter approximation to the marginal is important for powerful decoder-type models**: few people in the community are comfortable with the idea of KL annealing, though it's clearly useful in practice and has some reasonable heuristic arguments for it. However, Maddison et al. (2017) notes that the KL divergence in the particle filter method *does not collapse during training* as observed in (Bowman et al., 2015).

3. **Particle filters are can help approximate multimodal distributions**: Depending on our choice of resampling methods, particle filters are more efficient per-sample for this kind of optimization, and they are more likely to find multiple modes than get stuck at one.

References

Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014.

Samuel R Bowman, Luke Vilnis, Oriol Vinyals, Andrew M Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space. *arXiv preprint arXiv:1511.06349*, 2015.

Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015.

Tim Cooijmans, Nicolas Ballas, César Laurent, Çağlar Gülçehre, and Aaron Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in Neural Information Processing Systems*, pages 2199–2207, 2016.

Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015.

Rahul G Krishnan, Uri Shalit, and David Sontag. Structured inference networks for nonlinear state space models. In *AAAI*, pages 2101–2109, 2017.

Tuan Anh Le, Maximilian Igl, Tom Jin, Tom Rainforth, and Frank Wood. Auto-encoding sequential monte carlo. *arXiv preprint arXiv:1705.10306*, 2017.

Piji Li, Wai Lam, Lidong Bing, and Zihao Wang. Deep recurrent generative decoder for abstractive text summarization. *arXiv preprint arXiv:1708.00625*, 2017.

Chris J Maddison, Dieterich Lawson, George Tucker, Nicolas Heess, Mohammad Norouzi, Andriy Mnih, Arnaud Doucet, and Yee Whye Teh. Filtering variational objectives. *arXiv preprint arXiv:1705.09279*, 2017.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. Regularizing and optimizing lstm language models. *arXiv preprint arXiv:1708.02182*, 2017.

Christian A Naesseth, Scott W Linderman, Rajesh Ranganath, and David M Blei. Variational sequential monte carlo. *arXiv preprint arXiv:1705.11140*, 2017.

Casper Kaae Sønderby, Tapani Raiko, Lars Maaløe, Søren Kaae Sønderby, and Ole Winther. Ladder variational autoencoders. In *Advances in Neural Information Processing Systems*, pages 3738–3746, 2016.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: A high-rank rnn language model. *arXiv preprint arXiv:1711.03953*, 2017.

Biao Zhang, Deyi Xiong, Jinsong Su, Hong Duan, and Min Zhang. Variational neural machine translation. *arXiv preprint arXiv:1605.07869*, 2016.

Xun Zheng, Manzil Zaheer, Amr Ahmed, Yuan Wang, Eric P Xing, and Alexander J Smola. State space lstm models with particle mcmc inference. *arXiv preprint arXiv:1711.11179*, 2017.