New York University

Capstone Advanced Lab

Learning Latent Sequence Representations of Language

Author:
Jonay Gaël Trénous

Advisors:
Prof. Kyunghyun Cho &
Prof. Joan Bruna Estrach

May 16, 2017

Contents

1 Introduction

This project investigates the use of sequences as latent variables in deep generative models for language. Sequence-to-Sequence Neural Machine Translation has emerged as a dominant paradigm [6, 2]. With the addition of attention mechanisms, they have become the tool of choice in large production systems, and also in a number of other applications such as caption-generation [1, 9, 10]. These models follow a framework of an encoder which processes the source, and a discriminative decoder which generates a conditional language model. In this work, we aim to learn a generative model of language that combines the Encoder-Decoder Framework with a variable-length latent variable model.

The technique of Variational Autoencoders [3] allows to perform inference and maximum-likelihood estimation in generative models that are parametrized by complex functions such as deep neural networks. It has been used extensively in image modeling and dimensionality reduction. Here we will instead employ a variable length *latent language model*, capturing the varying information contents in natural language phrases.

Related Work has appeared recently in Miao and Blunsom [5], where they successfully trained a generative model for the task of sentence compression. In Zhang, Xiong, and Su [11], a variational model is used for machine translation. Their approach differs from this one in the structure of the graphical model: The latent variable is generated conditioned on the input and used as auxiliary input to the decoder, whereas we model the observation drawn conditioned on the latent variables

The report is structured in a mathematical description of the proposed model section 2, followed by the details of its parametrization and implementation section 3. In section 4, I present the preliminary (negative) experimental results and discuss directions to address the issues presented.

1.1 Notation

Throughout, I use bold face $\mathbf{x}, \mathbf{z}$ for sequences of random variables, and subscripts x_i, z_i for their elements. The letters x, z stand respectively for the the observed and latent variables. Upper case Letters are used to denote parametric functions (Neural Networks) as a function of their input $F(\mathbf{x}, z_i, ...)$.

2 Variational Inference in the Graphical Model

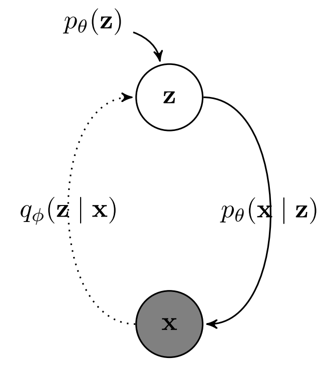
$p_\theta(\mathbf{z})$
$\mathbf{z}$
$q_\phi(\mathbf{z} \mid \mathbf{x})$
$p_\theta(\mathbf{x} \mid \mathbf{z})$
$\mathbf{x}$

Figure 1: The graphical Model

The graphical model underlying this work is depicted in Figure 1. A latent sequence $\mathbf{z}$ is drawn from the prior $p_\theta(\mathbf{z})$. The observed sequence $\mathbf{x}$ is then drawn conditioned on the latent variable. It is our goal to perform a maximum likelihood estimation of the parameters θ. Both $\mathbf{z}, \mathbf{x}$ live in sequence spaces. For ease of computation, the latent variable is modeled as a sequence of real numbers. The prior $p_\theta(\mathbf{z})$ factors over the length.

$$p_\theta(\mathbf{z}) = p_\theta(\text{len}(\mathbf{z})) \prod_{i=1}^{\text{len}(\mathbf{z})} p_\theta(\mathbf{z}_i)$$

The prior over the length is a truncated geometric distribution with hyperparameter $\gamma \in (0, 1)$, whereas the prior over the sequence given the length is a zero-mean stationary Gaussian process with identity covariance

$$p_\theta(l) \propto \gamma^n, \mathbf{z}_i \sim \mathcal{N}(0; 1)$$

The conditional distribution $p_\theta(\mathbf{x} \mid \mathbf{z})$ is modeled by an autoregressive RNN. For such a parametrization, both the posterior $p_\theta(\mathbf{z} \mid \mathbf{x})$ and the marginal $p_\theta(\mathbf{x})$ are intractable. We introduce a variational approximation of the posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$. Then we can write (See Kingma and Welling [3])

$$\log p_\theta(\mathbf{x}) = D_{KL}(p_\theta(\mathbf{x}) \mid \mathbf{z} \parallel q_\phi(\mathbf{x} \mid \mathbf{z})) + \mathcal{L}(\mathbf{x}, \theta, \phi)$$

As the first term on RHS is nonnegative, $\mathcal{L}$ gives us the *variational lower bound* to the loglikelihood function.

$$\mathcal{L}(\mathbf{x}, \theta, \phi) = \underbrace{-D_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}) \parallel p_\theta(\mathbf{z}))}_{\text{Normalizer}} + \underbrace{\mathop{\mathbb{E}}_{\mathbf{z} \sim q_\phi} p_\theta(\mathbf{x} \mid \mathbf{z})}_{\text{Reconstruction Term}}$$

The variational lower bound becomes tight when the approximate posterior is equal to the true almost everywhere. Its two terms have an interpretation as a normalizer that draws the approximate posterior close to the prior, and a reconstruction term that describes how well the approximate posterior describes the data. The lower bound can be efficiently approximated by Monte Carlo methods. If $\{\mathbf{z}^{(i)}\}_{i=1}^{M}$ is a set of i.i.d. samples from the approximate posterior, then the reconstruction term is approximately

$$\mathop{\mathbb{E}}_{\mathbf{z} \sim q_\phi} p_\theta(\mathbf{x} \mid \mathbf{z}) \simeq \frac{1}{M} \sum_{i=1}^{M} p_\theta(\mathbf{x} \mid \mathbf{z}^{(i)})$$

In practice, the reparametrization trick [3] is used to reduce the variance of this estimator. For Gaussian variables, the samples are drawn from a standard Gaussian, and then transformed using the location and scale parameters.

3 Implementation

This section gives a detailed outline of the parametrization of the described model. As mentioned in the introduction, we employ Recurrent Neural Networks to parametrize distributions over sequences. The model is trained by stochastic gradient ascent on the variational lower bound. Gradients with respect to most parameters can be computed using automatic differentiation. This is not possible with respect to the length of the latent sequence, the REINFORCE algorithm is used to estimate those gradients.

3.1 Prior $p_\theta(\mathbf{z})$

As mentioned before

$$p_\theta(l) \propto \gamma^n, z_i \sim \mathcal{N}(0; 1)$$

The hyperparameter γ determines the entropy of the latent space.

3.2 Parametrization of the approximate posterior

The variational distribution is parametrized in the following way:

- A bidirectional recurrent neural net $R_1(\mathbf{x})$ processes the source sequence and produces a sequence of state vectors $\mathbf{h}^{r,l}$

- A feedforward neural net $F_1(h_i^{r,l})$ produces scores α_i for each annotation. The weighted sum of the annotations is passed into another Feed Forward Net $F_2(\sum_i \alpha_i h_i^{r,l})$ which outputs an discrete probability vector $\pi, \pi_k = q_\phi(\text{len} = k \mid \mathbf{x})$ as the posterior approximation to the length distribution.

- A number $\hat{l} \sim \text{cat}(\pi)$ is drawn

- An autoregressive recurrent neural net $A_1(\mathbf{h}^{r,l}, \hat{l})$ with state vectors e_i produces the latent sequence $\mathbf{z}$ of length $\hat{l}$. At step i:

  - A feed forward neural net $F_3(h_j^{r,l}, e_{i-1})$ produces scores β_{ij} for each $h_j^{r,l}$. The weighted sum of the annotation is denoted as $c_i = \sum_{j=1}^{\text{len}(\mathbf{x})} \beta_{ij} h_j^{r,l}$.
  - The next state vector is computed as $e_i = a(e_{i-1}, z_{i-1}, c_i, \hat{l})$
  - Mean and variance parameters μ_i, σ_i are computed from the state vector in a feed forward neural net $F_4(e_i)$
  - A random variable $\epsilon_i \sim \mathcal{N}(0;1)$ is drawn
  - $z_i = \sigma_i \epsilon_i + \mu_i$

All the feedforward neural nets are one hidden layer with Tanh activations. This distribution factors in the same way as the prior:

$$q_\phi(\mathbf{z} \mid \mathbf{x}) = q_\phi(l \mid \mathbf{x}) \prod_{i=1}^{l} q_\phi(z_i | l, \mathbf{z}_{-i}, \mathbf{x})$$

3.3 Parametrization of the conditional $p_\theta(\mathbf{x} \mid \mathbf{z})$

This parametrization is fairly similar to the previous one, but differs as the distribution is over discrete sequences.

- A bidirectional recurrent neural net $R_2(\{z_i\})$ processes the latent sequence and produces state vectors $\hat{h}_i^{r,l}$

- An autoregressive recurrent neural net $A_2(\{\hat{h}_i^{r,l}\})$ with state vectors d_i predicts a target sequence $\hat{\mathbf{x}}$. At step i:

  - A feed forward neural net $F_5(\hat{h}_j^{r,l}, d_{i-1})$ produces scores γ_{ij} for each $\hat{h}_j^{r,l}$. The weighted sum of the annotation is denoted as $a_i = \sum_{i=1}^{k} \gamma_{ij} \hat{h}_j^{r,l}$.
  - The next state vector is computed as $d_i = f(d_{i-1}, \hat{x}_{i-1}, a_i)$
  - A probability vector π^i is computed in a one-layer feed forward neural net $F_6(d_i)$
  - A word $\hat{x}_i \sim \text{cat}(\pi^i)$ is drawn
  - if $\hat{x}_i = \texttt{STOP}$, end sequence, else repeat.

3.4 Computing the KL Divergence

The KL Divergence can be split into the discrete and continuous part:

$$
\begin{aligned}
&D_{KL}(q_\phi(z \mid x) \parallel p_\theta(z)) && (1)\\
&= \sum_{n=1}^{\max} q_\phi(l \mid x) \mathop{\mathbb{E}}_{q_\phi(z|x,l)} \log \frac{q_\phi(l \mid x) q_\phi(z \mid x, l)}{p_\theta(l) p_\theta(z \mid l)} && (2)\\
&= \sum_{n=1}^{\max} q_\phi(l \mid x) \left[\log \frac{q_\phi(l \mid x)}{p_\theta(l)} + \mathop{\mathbb{E}}_{q_\phi(z|x,l)} \log \frac{q_\phi(z \mid x, l)}{p_\theta(z \mid l)} \right] && (3)\\
&= \sum_{n=1}^{\max} q_\phi(l \mid x) \log \frac{q_\phi(l \mid x)}{p_\theta(l)} + \sum_{n=1}^{\max} q_\phi(l \mid x) \mathop{\mathbb{E}}_{q_\phi(z|x,l)} \log \frac{q_\phi(z \mid x, l)}{p_\theta(z \mid l)} && (4)\\
&= D_{KL}(q_\phi(l \mid x) \parallel p_\theta(l)) + \mathop{\mathbb{E}}_{q_\phi(l|x)} \left[D_{KL}(q_\phi(z \mid x, l) \parallel p_\theta(z \mid l)) \right] && (5)
\end{aligned}
$$

The first term can be computed exactly. The second term of eq. (5) is the KL Divergence of the prior and posterior of the latent Gaussian process. In the standard (non-sequential) variational autoencoder, the posterior is a multivariate Gaussian with identity covariance. That is not the case for this model, as the mean and variance μ_i, σ_i are functions of the previous draw z_{i-1} – and integrating over this function is intractable, ruling out a closed form solution for the KL divergence. Instead, we compute the KL divergence step by step using the sample that is drawn. For a sequence of length two, we can write

$$
\begin{aligned}
&D_{KL}(q_\phi(z_1, z_2) \parallel p_\theta(z_1, z_2))\\
&= \int \int q_\phi(z_1) q_\phi(z_2 \mid z_1) \log \frac{q_\phi(z_1) q_\phi(z_2 \mid z_1)}{p_\theta(z_1) p_\theta(z_2)} dz_2 dz_1\\
&= \int \int q_\phi(z_1) q_\phi(z_2 \mid z_1) \left[\log \frac{q_\phi(z_1)}{p_\theta(z_1)} + \log \frac{q_\phi(z_2 \mid z_1)}{p_\theta(z_2)} \right] dz_2 dz_1\\
&= \int \int q_\phi(z_1) q_\phi(z_2 \mid z_1) \log \frac{q_\phi(z_1)}{p_\theta(z_1)} dz_2 dz_1\\
&\quad + \int \int q_\phi(z_1) q_\phi(z_2 \mid z_1) \log \frac{q_\phi(z_2 \mid z_1)}{p_\theta(z_2)} dz_2 dz_1\\
&= \int q_\phi(z_1) \log \frac{q_\phi(z_1)}{p_\theta(z_1)} dz_1 + \int q_\phi(z_1) \int q_\phi(z_2 \mid z_1) \log \frac{q_\phi(z_2 \mid z_1)}{p_\theta(z_2)} dz_2 dz_1\\
&= D_{KL}(q_\phi(z_1) \parallel p_\theta(z_1)) + \mathop{\mathbb{E}}_{z_1} \left[D_{KL}(q_\phi(z_2 \mid z_1) \parallel p_\theta(z_2)) \right]
\end{aligned}
$$

Which shows that the sum of the KL divergences over each position gives an unbiased estimate of the full KLD.

$$
D_{KL}(q_\phi(\mathbf{z} \mid l) \parallel p_\theta(\mathbf{z} \mid l)) \simeq \sum_{i=1}^{l} D_{KL}(q_\phi(z_i \mid z_{-i}) \parallel p_\theta(z_i))
$$

A disadvantage is that the variance of this estimate increases with the length of the sequence.

3.5 The lower bound

We can now estimate the lower bound to the log likelihood

1. Sample $\mathbf{z}^{(i)} \sim q_\phi(\mathbf{z} \mid \mathbf{x})$
2. In the process of sampling $\mathbf{z}^{(i)} = \{z_1^i, ..., z_k^i\}$, store the parameters $\boldsymbol{\mu}^{(i)} = \{\mu_1^i, ..., \mu_k^i\}, \boldsymbol{\sigma}^{(i)} = \{\sigma_1^i, ..., \sigma_k^i\}$ of the distribution to estimate the KLD as the sum of KLDs over the sequence length

$$D_{KL}(q_\phi(\mathbf{z} \mid \mathbf{x}, l) \parallel p_\theta(\mathbf{z} \mid l)) \simeq \sum_{j=1}^{k} D_{KL}(\mathcal{N}(\mu_j^{(i)};\ \sigma_j^{(i)}) \parallel (\mathcal{N}(0;1)))$$

3. Compute $\log p_\theta(\mathbf{x} \mid \mathbf{z}^{(i)})$ by feeding the true target into the autoregressive RNN A_2:

$$\log p_\theta(\mathbf{x} \mid \mathbf{z}^{(i)}) = \sum_j \log \pi_{(i)}^j(x_j)$$

3.6 Training by Backpropagation

The described network admits gradients everywhere except in the discrete and stochastic length variable l. The length depends on a probability vector $\pi = \{\pi_j\}$, which is the output of a softmax unit with inputs $\{s_j\}$. The REINFORCE algorithm can be used to derive unbiased gradient estimates of the lower bound w.r.t each of these scores. For each $\mathbf{z}^{(i)}, \mathrm{len}(\mathbf{z}^{(i)}) = k^i$ that is drawn from $q_\phi(\mathbf{z} \mid \mathbf{x})$ denote its contribution to the lower bound estimate as

$$r_i = \log p_\theta(\mathbf{x} \mid \mathbf{z}^{(i)}) - \sum_{j=1}^{k^i} D_{KL}(\mathcal{N}(\mu_j^{(i)};\ \sigma_j^{(i)}) \parallel (\mathcal{N}(0;1)))$$

The REINFORCE algorithm guarantees that the following is an unbiased estimate of the gradients with respect to the input of the softmax unit Williams [8]:

$$\frac{\partial \mathcal{L}(\mathbf{x}, \theta, \phi)}{\partial s_j} \approxeq \lambda(r_i - \beta)\frac{\partial \log \pi_{k^i}}{\partial s_j}$$

Where λ is a balancing factor, π_k the probability of length $= k$. The reward baseline β significantly improves the speed of convergence over naive reinforcement [7]. The derivative of the log-softmax is given by

$$\frac{\partial \log \pi_{\hat{l}}}{\partial s_{\hat{l}}} = \frac{\pi_{\hat{l}} - \pi_{\hat{l}}^2}{\pi_{\hat{l}}} = 1 - \pi_{\hat{l}};\ \frac{\partial \pi_{\hat{l}}}{\partial s_i} = -\frac{\pi_i \pi_{\hat{l}}}{\pi_{\hat{l}}} = -\pi_i$$

A simple baseline is given by an exponential moving average of the reward with parameter $\alpha \in (0, 1)$. Denote by $r[t]$ the value of the reward at timestep t. Then

$$\beta[0] = r[0], \beta[t] = (1 - \alpha) * r[t] + \alpha * \beta[t-1]$$

Given this method of estimating gradients with respect to the parameters of the length distribution, the training procedure at each sample $\mathbf{x}$ can be summarized as following:

1. Do a forward pass through R_1, F_1, F_2 to compute the posterior $q_\phi(\mathbf{z}|\mathbf{x})$
2. For $i = 1$ to m do:
   (a) Sample $\hat{l}^i \sim q_\phi(l|x)$
   (b) Sample $\mathbf{z}^i \sim q_\phi(\mathbf{z}|\mathbf{x}, \hat{l}^i)$
   (c) Compute r_i
3. Compute the Monte-Carlo estimate of the objective

$$\mathcal{L}(\mathbf{x}, \phi, \theta) \approxeq \frac{1}{m} \sum r_i - D_{KL}(q_\phi(l \mid x) \parallel p_\theta(l))$$

   And backpropagate the gradients.

4. Use REINFORCE to estimate gradients of the rewards r w.r.t. the softmax scores s_j:

$$\frac{\partial r}{\partial s_j} \approxeq \frac{1}{m} \sum_{i=1}^{m} \lambda(r_i - \beta) \frac{\partial \log \pi_{\hat{l}^i}}{\partial s_j}$$

5. Backpropagate these gradients

3.7 Codebase

The code is built upon the PyTorch implementation of the OpenNMT framework [4]. The codebase proved very useful to build upon, providing an off-the-shelf implementation of the basic building blocks in terms of Encoder and Decoder with Attention. Most of the additions were related to variational loss and the Reinforcement signal.

4 Preliminary Results and Analysis

So far, the preliminary results are *negative*. There were two observed behaviors during training:

- The model entirely disregards the latent representation, as seen by the KL divergence going to zero almost immediately (see Figure 2). In this case, the decoder learns a language model for the target sentences, but no signal from the encoding is used.
- When setting low weight to the KL divergence in the loss, the model is able to learn an identity mapping in the latent space. But when increasing the weight of the KL divergence, it reverts to the first trivial local minimum.

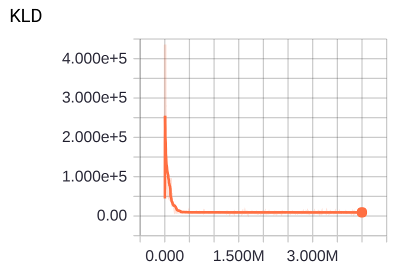

Figure 2: The typical behavior of KL Divergence during training

Clearly, both of these mappings are not interesting. It is curious that the model goes from the identity mapping directly to the other extreme, without finding an intermediate value. These findings indicate that the computation of the KL divergence is the reason for the failure of the current model. I believe it is likely that there are still coding errors. It is also possible that the current KL divergence estimator suffers from a high variance. At this point, there is a need for more experiments and code review to give a satisfactory answer to this question.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural Machine Translation by Jointly Learning to Align and Translate". In: *CoRR* abs/1409.0473 (2014). URL: `http://arxiv.org/abs/1409.0473`.

[2] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation". In: *CoRR* abs/1406.1078 (2014). URL: `http://arxiv.org/abs/1406.1078`.

[3] D. P Kingma and M. Welling. "Auto-Encoding Variational Bayes". In: *ArXiv e-prints* (Dec. 2013).

[4] G. Klein et al. "OpenNMT: Open-Source Toolkit for Neural Machine Translation". In: *ArXiv e-prints* (). eprint: `1701.02810`.

[5] Yishu Miao and Phil Blunsom. "Language as a Latent Variable: Discrete Generative Models for Sentence Compression". In: *CoRR* abs/1609.07317 (2016). URL: `http://arxiv.org/abs/1609.07317`.

[6] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to Sequence Learning with Neural Networks". In: *Advances in Neural Information Processing Systems 27*. Ed. by Z. Ghahramani et al. Curran Associates, Inc., 2014, pp. 3104–3112. URL: `http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf`.

[7] Richard S Sutton et al. "Policy gradient methods for reinforcement learning with function approximation." In: *NIPS*. Vol. 99. 1999, pp. 1057–1063.

[8] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3-4 (1992), pp. 229–256.

[9] Yonghui Wu et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation". In: *arXiv preprint arXiv:1609.08144* (2016).

[10] Kelvin Xu et al. "Show, attend and tell: Neural image caption generation with visual attention". In: *International Conference on Machine Learning*. 2015, pp. 2048–2057.

[11] Biao Zhang, Deyi Xiong, and Jinsong Su. "Variational Neural Machine Translation". In: *CoRR* abs/1605.07869 (2016). URL: `http://arxiv.org/abs/1605.07869`.