A Comparison of RNN-based Deep Language Models

Jake Daly
University of California, San Diego
CSE 291 - Advanced Statistical NLP
PID: A53312764
jmdaly@eng.ucsd.edu

ABSTRACT

In this project we will use PyTorch to train several different flavors of RNN-based deep language models, demonstrating the merits, drawbacks, and challenges that each model faces. The first model, a vanilla RNN-LM, will be trained on the Penn Tree Bank (PTB) dataset and uses the log data likelihood as its training objective. We will train the second model, a variational autoencoder (VAE), on the same dataset using two different RNN architectures: a traditional RNN and a GRU. Despite being state-of-the-art language modelling techniques, VAEs face several that we explore solutions to, including KL-annealing and free bits (KL thresholding). In the final part of this report, we apply the variational autoencoder to a Twitter dataset (tweets from politicians) and explore what sort of information is encoded into the latent space after training.

1 INTRODUCTION

Last project we trained a traditional Ngram-based language model using interpolated Kneser-Ney trigram modeling. In this project, we set out with the same objective: to maximize the likelihood of the observed data, using previous words as context for the distribution on the next word. When trying to achieve this goal with the Ngram model, we came across a few different obstacles which we will see RNNs are inherently well poised to solve.

First, the Ngram model was not flexible in the number of words it could consider in its context: for the trigram model, we were forced to always consider two words as the context for the next word. This rigid structure is a pain at the beginning of sentences when we don't have any context, and is also inconvenient in the middle and end of sentences because it ignores out-of-context words that might have semantic value. RNN's inherently overcome this with their ability to recurrently accept new data at each time step and thus allow contexts of varying lengths.

The other big issue that arose in the Ngram model was sparsity, which grew linearly with the size of the vocabulary and combinatorially with the order of the Ngram. This is because Ngram models rely on counts of words and specific word contexts, and hence we cannot use word embeddings to cluster words with similar semantics. In contrast when we model using RNN-based techniques, we can use pre-trained word embeddings to embed words with similar semantics close to each other.

Aside from using RNNs to solve some of the big issues that Ngram models have, we can also use them as a vehicle introduce latent variables to learn high level information about the data source we are modeling. These variational autoencoders can capture tone, topic, sentiment, and many other types of information that can be viewed as having an influence on how the observed data was produced. We will see that training these VAEs comes with it's own set of challenges, and we will explore several techniques for overcoming them.

The rest of this paper is organized as follows:

- **Section 2:** Intuition and theory behind RNN-based LMs, the variational autoencoder, and the issue of posterior collapse
- **Section 3:** Dynamics of training the VAE, amortized inference, and measuring performance
- **Section 4:** Using the KL-annealing and free bits techniques to protect the model from posterior collapse
- **Section 5:** Exploration of the type of information that the latent space can encode using a novel Twitter dataset.

2 THEORY OF DEEP LATENT VARIABLE MODELS

The application of RNN models to language processing comes very naturally from the temporal and sequential characteristics of language. The state of the model is a function of some input x_t and the model's hidden state h_{t-1} from the previous iteration. The output of the model h_t is used as feedback into the model during the next iteration. This is summarized in Figure 1.

VAEs [2] extend this idea by introducing a latent space z to encode information about how the data was generated. As we are training our model, we treat the parameters of the encoding network as part of the training objective, and z is then able to learn information about the data it is being trained on. This x and z are illustrated in Figure 4.

2.1 Derivation of objective functions and training

With the newly introduced "influencing" variable z, the new MLE estimator can be calculated from the joint distribution $P(x, z)$ by

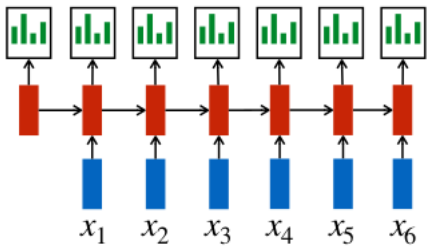

Figure 1: An unrolled depiction of an RNN language model [1]

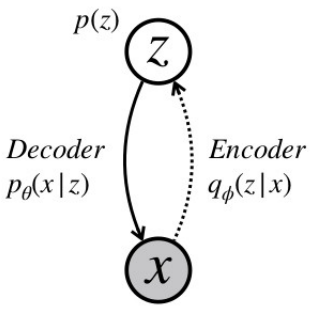

Figure 2: The latent variable z depicted as a DAG. Intuitively we think of the variable z as having an influence on how the observed data x is generated. [1]

marginalizing x over z

$$P(x) = \int P(x,z)dz = \int P(z) \cdot P(x|z)dz$$

In practice these probabilities are incredibly small, so we take the log of this to prevent underflow, arriving at $log\ p_\theta(x)$ (where theta represents model parameters).

This marginal distribution over z requires integrating over a combinatorially large latent space, which is computationally intractable. Fortunately, we are able to derive a lower bound (called the evidence lower-bound, or ELBO) on the true log likelihood of the data by factoring this expression into two terms: the reconstruction loss of the decoder, and the Kullback-Leibler divergence between the latent space z (a mixture of Guassian distributions) and a learned encoder.

Below we show how ELBO is derived, and afterwards we discuss important details that pertain to using ELBO as the training objective.

$$\begin{aligned}
log\ p_\theta(x) &= log \int p_\theta(x,z)\ dz \\
&= log \int p_\theta(z)p_\theta(x|z)\ dz \\
&= log \int p_\theta(z)p_\theta(x|z)\ dz \\
&= log \int p_\theta(z)p_\theta(x|z) \cdot \frac{q_\phi(z|x)}{q_\phi(z|x)}\ dz \\
&= log\, \mathbb{E}_{q_\phi(z|x)}[\frac{p_\theta(z)\ p_\theta(x|z)}{q_\phi(z|x)}] \\
&\geq \mathbb{E}_{q_\phi(z|x)}[log\ \frac{p_\theta(z)\ p_\theta(x|z)}{q_\phi(z|x)}] \\
&= \mathbb{E}_{q_\phi(z|x)}[log\ p_\theta(x|z)] + \mathbb{E}_{q_\phi(z|x)}[log\ \frac{p_\theta(z)}{q_\phi(z|x)}] \\
&= \mathbb{E}_{q_\phi(z|x)}[log\ p_\theta(x|z)] + \int log\ q(z|x) \cdot \frac{p_\theta(z)}{q_\phi(z|x)}dz \\
&= \mathbb{E}_{q_\phi(z|x)}[log\ p_\theta(x|z)] - KL(q_\phi(z|x)\ ||\ p_\theta(z))
\end{aligned}$$

To train the VAE according to this objective, we first pass the training batch x through the encoder to obtain a hidden tensor h. The hidden tensor is then mapped to the latent space z by a linear transformation, and its mapped elements μ_i are used to perform a multivariate Gaussian sample. This multivariate Gaussian serves as the conditioning variable in the decoder when we are reconstructing x. To perform this reconstruction, we map z back to the vector space which contains the hidden vectors h via another linear transformation, and then we send this vector through the decoder RNN. The linear transformations and RNN model parameters constitute the VAE's parameters to be trained, as they all make up different parts of the variational approximation to the model's log likelihood.

One issue that often arises when training VAEs, called **posterior collapse**, occurs when the KL divergence term in ELBO collapses to 0 (or near 0). Posterior collapse can be thought of as a local optimum, where the model's generator and discriminator have learned to "agree" with each other: $p_\theta(z|x) = q_\phi(z|x) = p(z)$. When this happens, it's as if the latent variable z does not exist, and the model is just a traditional RNN without any latent information. More on this in Section 4.

3 TRAINING DYNAMICS & MEASURING PERFORMANCE

Being highly parameterized models, RNNs suffer from the issue of tuning hyperparameters to optimal values. Whereas with other machine learning models it is reasonable to perform a grid search to tune hyperparameters, this is not the case with RNNs, because they not only have many parameters to tune, but the bigger and deeper they are, the slower it is to train and search for these parameters.

In our case, an exhaustive search was not possible due to resource and time limitations of the experiment. Instead of performing a grid search, we perform a random search over the ranges of each variable that seem reasonable, choose the parameters from the "best" run ("best" is to be defined in the next section), and then perform a localized grid search around those parameters.

To further accelerate the hyperparameter tuning stage, GPU's from Google's Colab project are used, which reduce the time of training from almost 3 hours per trial (CPU), to just over 10 minutes (GPU). Performing the above "random to localized grid searches" for all experiment types would not have otherwise been feasible. The resulting sets of best hyperparameters from the random-to-grid searches are shown in Table 1.

3.1 ELBO as a performance metric

In Section 2 we established a lower-bound (ELBO) on the model's actual log likelihood $P(x)$. Using ELBO as an approximation to allow amortized inference leads to an obvious issue: we don't know the true loss of our model. This makes it hard to compare the performance against other models that use an exact loss function, such as the traditional RNN. The best-case scenario for comparing the log likelihood of a VAE to that of a traditional RNN is if the ELBO of the VAE is larger than the log likelihood of the RNN, because we know the actual log likelihood of the VAE will be at least as large as its ELBO (since it's a lower bound), and would thus be higher than the log likelihood of the RNN.

Table 1: Hyperparameters that resulted in the best (lowest) loss for each model class.

Model Type	hs	ls	nl	ed	wd	eb	rnn	klt	KLL (T)	Loss (T)	KLL (V)	Loss (V)
RNN	256	16	1	.5	0	300	gru	N/A	**N/A**	**74.53**	**N/A**	**116.59**
VAE-RNN	1024	16	1	.45	.05	350	rnn	1	**6.4**	**75.16**	**6.4**	**112.68**
VAE-GRU	272	8	2	.55	.01	350	gru	6	**6.33**	**70.91**	**9.72**	**107.79**

Another thing to be cautious of when evaluating the performance of the RNN is the KL divergence term in the ELBO. If it has collapsed, it's an indicator that the model has learned to ignore the latent variable z (in effect, the model is essentially just an RNN). In selecting one of the sets of hyperparameters during the train runs, we want to select those associated with

(1) as low of an ELBO score as possible
(2) a larger (but not too large) KL divergence
(3) (in an ideal world) smaller network dimensions

With these caveats in mind, we can assess the effectiveness of the models, whose hyperparameters and resulting losses are shown in Table 1. The columns from left to right are: hidden size, latent size, number of layers, embedding dropout, word dropout, embedding size, architecture, KL-threshold (described in the next section), KL divergence on training data, Loss (or ELBO) on training data, KL divergence on validation data, Loss (or ELBO) on validation data.

4 IMPROVING THE VAE

To deal with the issue of posterior collapse, we use two separate techniques that each have different independent effects on the loss function. The first technique, called KL annealing, involves multiplying the KL term by a constant, so that the new loss function is given by:

$$\mathbb{E}_{q_\phi(z|x)}[log\ p_\theta(x|z)] - \alpha(t) \cdot KL(q_\phi(z|x)\ ||\ p_\theta(z))$$

This weighted KL term allows us to choose how much the KL divergence plays a roll in the overall loss function. We can set this $\alpha(t)$ to mute the amount that the KL divergence contributes at the beginning of training so that the optimizer can focus at first on minimizing the reconstruction loss, discouraging the KL term from collapsing.

The second modification we make is using the free bits (also known as "KL thresholding") technique, in which the KL divergence is prevented from dropping below a certain threshold by taking $max(KL, \epsilon)$, where epsilon is a hyperparameter of the model. In our experiments, we apply this *max* function to the entire KL term, rather than on every dimension of KL. This has a similar effect to annealing in that it causes the optimizer to not focus on KL once the term is equal to a constant. This produces a zero-gradient, and the encoder parameters which compose $q_\phi(z|x)$ effectively stop receiving updates during backwards pass, so the optimizer only works on minimizing the reconstruction loss.

As Table 1 shows, both techniques are effective are preventing posterior collapse.

5 BEYOND PTB: TWEETS FROM POLITICIANS

In this section we go a bit further, training a new VAE from scratch on an entirely new dataset and exploring the features learned by the latent space. The dataset used in this part of the experiment are tweets from American politicians, who have been categorized as left (Democrats) and right (Republicans) by their political party affiliations:

- **Democrats**: Bernie Sanders, Elizabeth Warren, Alexandria Ocasio-Cortez, Hillary Clinton, Dick Durbin, Joe Manchin, Jon Tester, Chuck Schumer, Christina Bellantoni, Donna Brazile, Glenn Greenwald, Nico Pitney
- **Republicans**: Mitch McConnell, Lindsey Graham, Donald Trump, Mitt Romney, Kevin McCarthy, Erick Erickson, Mindy Finn, Tim Carney, Ana Marie Cox, Tucker Carlson

After using the Twitter API to create new train, valid, and test dataset splits, we proceed to train the model as described before, first using a random-search, then performing a more localized grid search on the best set of hyperparameters. Figure 4 shows ELBO of the models during this search phase versus the final validation KL divergence and reconstruction losses. The red data points show the results from the random search; the blue points show the results from the localized grid search. This figure makes apparent all of

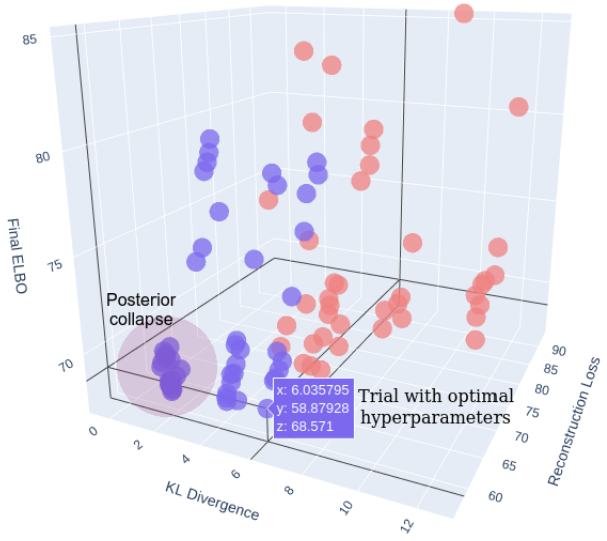

Figure 3: Hyperparameter search using the Twitter dataset. Red points are the training results from the random search; purple points are a grid search around the hyperparameters obtained from the random search's best run.

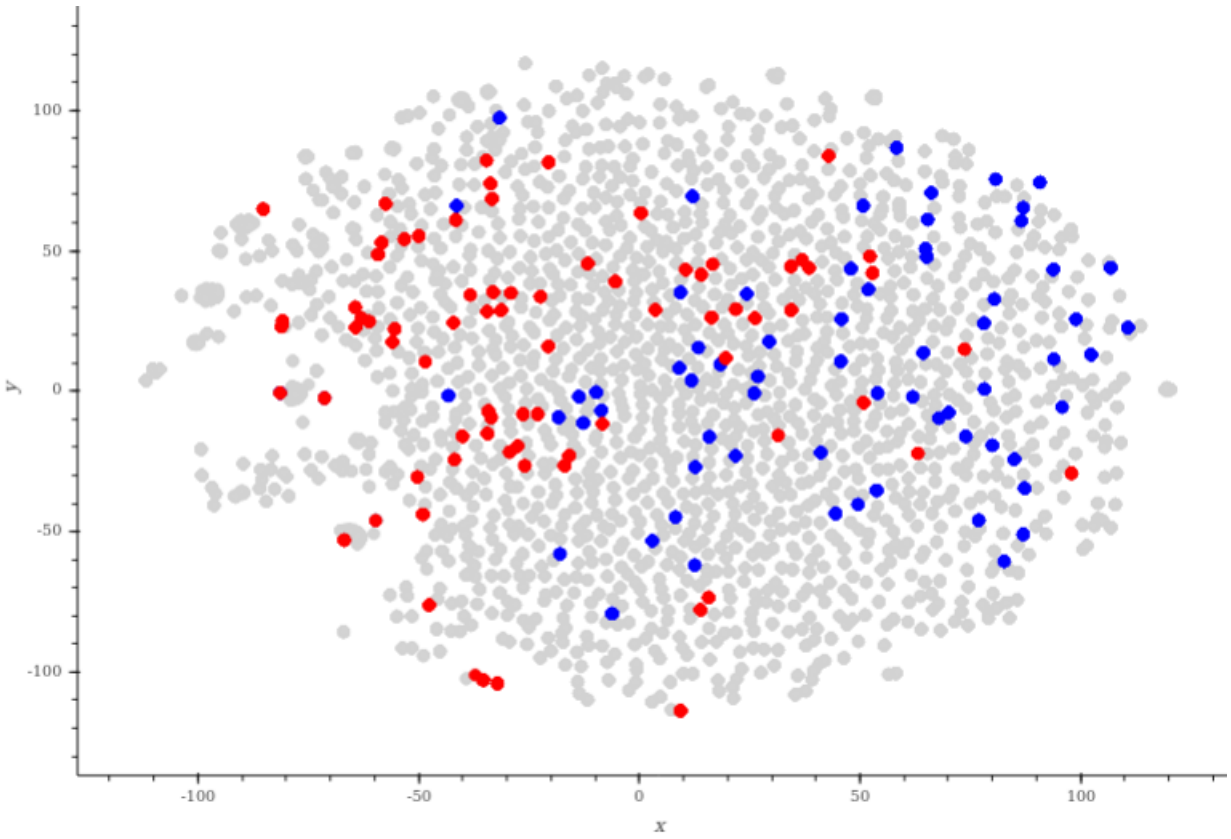

Figure 4: Several tweets from Mitch McConnell (red) and Alexandria Ocasio-Cortez (blue), encoded into the latent space z, then embedded in $\mathbb{R}^2$ via a t-SNE transformation

the models that experienced posterior collapse, for example the clump of blue data points in the lower left who's KL divergence is approximately 0. We can choose a point from this set which has a low ELBO and reconstruction loss, while having a KL divergence > 1.

We would expect the model to capture latent information about the dataset, such as political affiliation, sentiment about political topics, and much more. Because the resulting hyperparameter set that was used for this model had a latent space of dimension 16, it's difficult to find structure in it. We attempt to find structure by using a non-linear t-SNE embedding (provided by the scikit-learn Python package) to map samples from this latent space to $\mathbb{R}^2$. A sample of a pair of these politicians is shown in the plot above.

The red and blue data represent one single comparison which appears to have a trend: Republicans (red) appear to the left, and Democrats (blue) appear to the right. This trend is not as strong when visualizing other pairs of politicians, which could be due to several reasons. First, we are embedding vectors from a 16-dimensional space into a 2-dimensional space, so any higher dimensional and non-linear structure that existed in 16 dimensions is corrupted in the 2-dimensional embedding. Furthermore, the t-SNE representation is well-known to more accurately model data points who are in proximity to each other, and to struggle displaying points that are further away from one another. On top of this, political left-ness or right-ness is a complex concept, and is just one of *many* features that the model could have learned. It's more than likely that the model learned simpler features from the dataset, rather than the complex notion of political affiliation. We can still be confident that our model *did* learn something, however, because despite the KL threshold being set to 4, the model's final KL was 8.4 (indicating that no posterior collapse occurred).

REFERENCES

[1] Taylor Berg-Kirkpatrick. *Variational Autencoders*, CSE-291: Advanced Statistical Natural Language Processing. University of California, San Diego. Spring 2020.
[2] Diederik P. Kingma, Max Welling. *Auto-Encoding Variational Bayes*. Universiteit van Amsterdam. 2014. [*On the electrodynamics of moving bodies*]. Annalen der Physik, 322(10):891–921, 1905.
[3] Zichao Yang, Zhiting Hu, Ruslan Salakhutdinov, Taylor Berg-Kirkpatrick. *Improved Variational Autoencoders for Text Modeling Using Dilated Convolutions*. Carnegie Mellon University. 2017.
[4] Junxian He, Daniel Spokoyny, Graham Neubig, Taylor Berg-Kirkpatrick. *Laging Inference Networks and Posterior Collapse in Variational Autoencoders*. ICLR. 2019.