State Space LSTM for NLP

T. Kirscher, Y. Remmache, L. Morisset

ENSAE - IP Paris

Januray 12, 2024

Introduction

- Significance of LSTMs in sequence modeling
  - LSTMs' ability to capture long-range dependencies
  - Applications in natural language processing (NLP)
- Interpretability advantages of state space models
- **Motivation**: Integrating interpretability of state space models with the power of LSTMs
- **Reference Paper**: [1] *State Space LSTM Models with Particle MCMC Inference*, Xun Zheng et al., CoRR, 2017

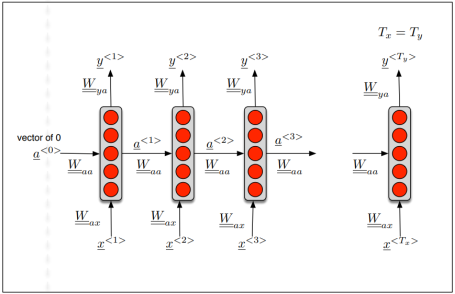
Overview

Table of content

Reccurent Neural Network: a type of neural network designed to process sequential data

Forward propagation (one hidden layer):

$$s_t = g_1(W_{ss}s_{t-1} + W_{sx}x_t + b_s)$$
$$\hat{y}_t = g_2(W_{ys}s_t + b_y)$$



Back Propagation Through Time (BPTT) and Vanishing Gradient

To train an RNN we need to back-propagate through layers and through time.

$$\frac{\partial \mathcal{L}}{\partial W_{ss}} = \sum_{t=1}^{t} \frac{\partial \mathcal{L}^{(t)}}{\partial W_{ss}}$$

$$\frac{\partial \mathcal{L}^{(t)}}{\partial W_{ss}} = \sum_{k=0}^{t} \left(\prod_{i=k+1}^{t} \frac{\partial s_i}{\partial s_{i-1}} \right) \frac{\partial s_k}{\partial W_{ss}}$$

Vanishing gradient appends when $|\frac{\partial s_i}{\partial s_{i-1}}| < 1$:

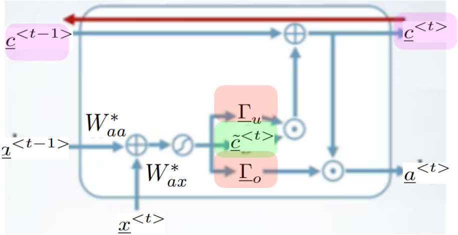
- Contributions from faraway steps vanish and don't affect the training
- Difficult to learn long-range dependencies

LSTM: a type of RNN introduced to learn long-term dependencies

Simplified LSTM:

We add a new path: the **memory cell** c_t

- The candidate cell value $\tilde{c}_t$ is added to the memory cell if the update gate Γ_u is open
- The hidden state is output if the output gate Γ_o is open



LSTM vs. SSM

Model	Pros	Cons
LSTM	Capture long-term dependencies	Not interpretable
SSM	Interpretable	Cannot capture long-term dependencies due to the Markovian nature of the latent variable process

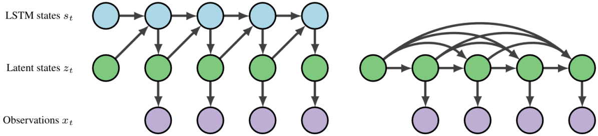
Table: Comparison of LSTM and SSM

SSL: a combination of SSM and LSTM models

Main idea: we place an LSTM on the latent space to simulate the states z_t. We then generate the observations x_t from a SSM.

Generative process:

1. $s_t = LSTM(s_{t-1}, z_{t-1})$ (LSTM part)
2. $z_t | z_{1:t-1} \sim p_\omega(z_t | z_{1:t-1}) =: p_\omega(z; g(s_t))$ (LSTM part)
3. $x_t | z_t \sim p_\phi(x; h(z_t))$ (SSM part)



(a) Full model with explicit LSTM states. (b) Model after collapsing LSTM states.

Figure 1: Generative process of SSL.

Background

Topical SSL:

- $x = (x_1, \ldots, x_t, \ldots, x_T)$ is a text sequence of length T
- $z = (z_1, \ldots, z_t, \ldots, z_T)$ is a sequence of unobserved topics characterizing the words in x

Example:

- $x =$ ("I", "was", "born", "in", "Palaiseau", "in", "1999")
- $z =$ ("Pronoun", "Verb", "Verb", "Preposition", "City", "Preposition", "Date")

Goal: create a generative model. Given a sequence $(x_1, \ldots, x_t)$, we want to predict $(x_{t+1}, \ldots, x_T)$.

Generative Model

x_t and z_t are discrete variables:

- $x_t \in \{1, \ldots, N\}$
- $z_t \in \{1, \ldots, K\}$

Where N is the size of the vocabulary and K is the number of topics. The SSL model can be written as:

1. $z_t | z_{1:t-1} \sim \text{Categorical}(\text{softmax}(Ws_t + b))$ $(LSTM)$
2. $x_t | z_t \sim \text{Categorical}(\phi_{z_t})$ (SSM)

Where:

- $\text{softmax}(Ws_t + b) = [p_\omega(z_t = 1 | z_{1:t-1}), \ldots, p_\omega(z_t = K | z_{1:t-1})]'$
- $\phi_{z_t} = [p_\phi(x_t = 1 | z_t), \ldots, p_\phi(x_t = N | z_t)]'$

Dataset used, pre processing and technical aspects

Dataset:

- IMBD dataset [2] : 25,000 movie reviews, keep $n = 200$ reviews
- Consider the 5000 "top words": $N = 5000$
- Truncation of the sequences: $T = 170$
  - $T_{\text{train}} = 100$
  - $T_{\text{test}} = 70$

LSTM:

- 64 hidden neurons
- Add a dropout layer with $p = 0.2$
- Optimizer: Adam
- z_t variables are one hot encoded in LSTM part

Particle Gibbs: a way to sample from the variational distribution in EM algorithm

Main idea:

We combine stochastic EM algorithm with backpropagation to train the SSL model. For each text and at each iteration of the EM algorithm, we sample $z^*_{1:T_{\text{train}}}$ from the optimal variational distribution in order to compute the MLE of the SSM and the LSTM. To sample $z^*_{1:T_{\text{train}}}$ we use an algorithm called **Particle Gibbs**.

Particle Gibbs Inference

Overview

- Sequential Monte Carlo (SMC) method
- Particle Gibbs inference for sampling from joint posterior
- No factorization assumptions

Particle Gibbs Inference

Algorithm

Algorithm Inference with Particle Gibbs

Require: P: number of particules, T: length of the sequence

Initialize $z_0^p = z_0$ and $\alpha_0^p = \frac{1}{P}$ for $p = 1, \dots, P$
for $t = 1, \dots, T$ **do**
    Fix reference path: set $a_{t-1}^1 = 1$ and $z_{1:t}^1 = z_{1:t}^*$ from the previous iteration
    for $p = 2, \dots, P$ **do**
        Sample ancestors $a_{t-1}^p \sim \alpha_{t-1}$
    end for
    for $p = 2, \dots, P$ **do**
        Sample particles $z_t^p \sim \gamma_t^p$ and set $z_{1:t}^p = (z_{1:t-1}^{a_{t-1}^p}, z_t^p)$
    end for
    for $p = 1, \dots, P$ **do**
        Compute normalized weights α_t^p
    end for
end for
Sample $r \sim \alpha_T$
return the particle path $z_{1:T}^{a_T^r}$

Particle Gibbs Inference

Recall:

- $\text{softmax}(Ws_t + b) = [p_\omega(z_t = 1|z_{1:t-1}), \ldots, p_\omega(z_t = K|z_{1:t-1})]'$
- $\phi_{z_t} = [p_\phi(x_t = 1|z_t), \ldots, p_\phi(x_t = N|z_t)]'$

Let $\phi = [\phi_1, \ldots, \phi_K] \in \mathbb{R}^{N \times K}$, such as $\phi[i,j] = p_\phi(x_t = i|z_t = j)$. Here:

- $\alpha_t = p(x_t|z_{1:t-1}) \propto \sum_{k=1}^{K} \text{softmax}(Ws_t + b)[k] \odot \phi_k$: unormalized distribution on $\{1, \ldots, N\}$
- $\gamma_t = p(z_t|z_{1:t-1}, x_t) \propto \text{softmax}(Ws_t + b) \odot \phi[x_t, 1:K]$: unormalized distribution on $\{1, \ldots, K\}$

Training Loop

Algorithm Training Loop for Topical SSL

Require: S_x: set of training sequences, P: number of particules, K: number of topics, N: size of the vocabulary, n: number of training samples, n_epochs: number of epochs

```
for i = 1, ..., n do
    z*_{1:T_train} ~ i.i.d U{1, ..., K}
end for
for epoch = 1, ..., n_epochs do
    S_{z*_{1:T_train}} = [ ]
    for i = 1, ..., n do
        Compute z*_{1:T_train} by Particle Gibbs using previous z*_{1:T_train}
        Add z*_{1:T_train} to S_{z*_{1:T_train}}
        Train LSTM with z*_{1:T_train} by back-propagation
        Compute the MLE of the SSM on S_x[1 : i] and S_{z*_{1:T_train}}[1 : i]
    end for
end for
```

- **for** $i = 1, \ldots, n$ **do**
  - $z^*_{1:T_{\text{train}}} \sim$ i.i.d $\mathcal{U}\{1, \ldots, K\}$
- **end for**
- **for** epoch $= 1, \ldots,$ n_epochs **do**
  - $S_{z^*_{1:T_{\text{train}}}} = [\,]$
  - **for** $i = 1, \ldots, n$ **do**
    - Compute $z^*_{1:T_{\text{train}}}$ by Particle Gibbs using previous $z^*_{1:T_{\text{train}}}$
    - Add $z^*_{1:T_{\text{train}}}$ to $S_{z^*_{1:T_{\text{train}}}}$
    - Train LSTM with $z^*_{1:T_{\text{train}}}$ by back-propagation
    - Compute the MLE of the SSM on $S_x[1:i]$ and $S_{z^*_{1:T_{\text{train}}}}[1:i]$
  - **end for**
- **end for**

Implementation Details

- All the algorithms coded in `pytorch`
    - GPU usage for LSTM
    - CPU usage for other parts
- n_epochs $= 5$
- $P = 10$
- $K \in \{10, 50, 100\}$
- Evaluation metric: perplexity

Perplexity curve for the LSTM model

Baseline

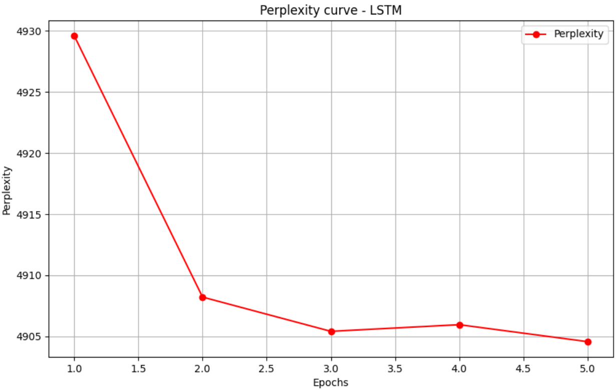
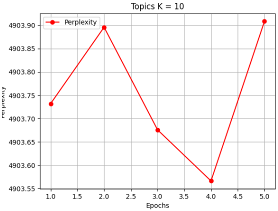
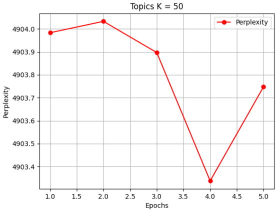


Figure: Perplexity through epochs for standard LSTM

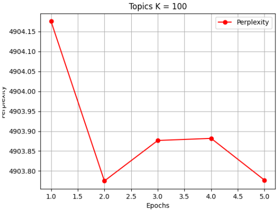
Perplexity curves for the SSL model



10 Topics



50 Topics



100 Topics

Topics Visualization (Interpretability)

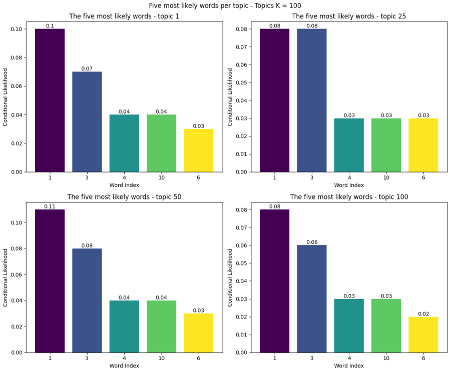


Figure: Latent Topics Visualization

Key insight: Conditionally on topic 50, the third word in the vocabulary is the second word most likely to appear, with a conditional probability of 0.08.

Conclusion: pros and cons of SSL

Pros:

- Similar performances to the LSTM model on our dataset. In the article, LSTM provides better results, but the amount of data is much greater
- Results are interpretable (main advantage)

Cons:

- Computational complexity: assuming that all internal operations can be performed in constant time (which is not the case), the algorithmic complexity of the Particle Gibbs algorithm is $\mathcal{O}(P + T(P-1)^2 P)$ for a single observation
- Concretely, with the same set-up: about 5 minutes to run one epoch with LSTM versus about 4 hours with SSL
- Choice of K: K too high can lead to overfitting whereas K too low can lead to underfitting

References

Xun Zheng, Manzil Zaheer, Amr Ahmed, Yuan Wang, Eric P. Xing, and Alexander J. Smola.
State space LSTM models with particle MCMC inference.
CoRR, abs/1711.11179, 2017.

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts.
Learning word vectors for sentiment analysis.
In Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

Q & A

Thank you!
Questions?