A Linear Dynamical System Model For Text

David Belanger (UMass Amherst)
Sham Kakade (Microsoft Research)
Semi-Supervised Learning in NLP

1. Do unsupervised learning that induces some reduced-dimensionality representation of text.

2. Apply the dimensionality reduction to annotated data.

3. Do supervised learning on the mapped data.
Word Embeddings

Map each word to a low dimensional vector

(Bengio et al. 2003; Mikolov et al., 2013; ...)

Country and Capital Vectors Projected by PCA
Word Tokens vs. Word Types

Types:
  What you look up in the dictionary.

Tokens:
  Words in context.

“The dog ran.”

Word embeddings are typically at the type level
Our Work
Goal:
Token Embeddings

We should embed word *tokens* in context.


“chair of the department” vs. “chair at the dinner table”
Consideration:
Advantages of Type-Level Training

Computational:
Training data compressed to sparse co-occurrence counts.
Size of matrix is independent of size of corpus!

Statistical:
\[ P(w) = \frac{\#(w)}{N} \quad \Rightarrow \quad P(w) = \frac{\#(w + \alpha)}{N + \alpha V} \]
Smoothing is difficult in token-level training.
Consideration: Latent-Variable Sequence Modeling

• Many word embedding methods consider sliding windows or bags of words.

• Text is structured as a sequence. Ideally our token embedding method would model this structure.

• The latent state yields dimensionality reduction
Our General Method

1) Learn a generative model for text sequences with a vector-valued latent variable for every token.

2) At test time, obtain token embeddings using posterior inference over these latent variables.
Related Work
• Latent-state sequence models trained at token level:
  – HMMs w/ Baum-Welch (Rabiner, 1986)
  – RNN language model (Mikolov et al., 2010)
  – Neural language model (Bengio et al., 2003)

• Sequence model with type-level training, but no dimensionality reduction:
  – Ngram language models

• Type-level training of word embeddings, but not a sequence model:
  – Glove (Pennington, et al., 2014)
  – PPMI factorization (Levy and Goldberg, 2014)
  – CCA (Dhillon et al., 2012, Stratos et al. 2015)

• Token-level training, but not a sequence model:
  – Word2Vec (Mikolov et al., 2013) and variants

• Type-level training of sequence model, but requires third-order statistics:
  – Spectral learning of HMMs (Hsu et al., 2008)
Linear Dynamical Systems
Gaussian Linear Dynamical System

Generative model:

\[ x_t = Ax_{t-1} + \eta \]
\[ w_t = Cx_t + \epsilon, \]

\[ \epsilon \sim N(0, D), \eta \sim N(0, Q) \]
Kalman Filter

- *Exact, Efficient* posterior inference for latent states.
- Maintains mean and variance for every timestep.
- Cubic in relevant dimensions.
- Forward and backward passes.
Steady State Kalman Filter

**Fact 1:**
The Kalman filter’s update to the posterior variances doesn’t depend on the actual observations.

**Fact 2:**
This variance reaches a steady state value quickly.

**Exact Kalman Filter**

\[
\begin{align*}
\hat{x}_t &= A\hat{x}_{t-1} \\
S_t^{-1} &= AS_{t-1}^{-1}A^\top + Q \\
K_t &= S_t^{-1}C(S_{t-1}^{-1}C^\top + D)^{-1} \\
\hat{x}_t &= \hat{x}_{t-1} + K_t(y_t - C\hat{x}_{t-1}^\top) \\
S_t &= S_{t-1}^{-1} - K_tCS_t^{-1} \\
\end{align*}
\]

**Kalman Filter w/ Steady State Assumption**

\[
\hat{x}_t = (A - K_{ss}CA)\hat{x}_{t-1}^{-1} + K_{ss}w_t
\]
Steady-State Filtering

\[ \hat{x}_t = (A - K_{ss}CA)\hat{x}_{t-1} + K_{ss}w_t \]

Precompute

Posterior mean at t-1, given observations including t-1.

Kalman Gain Matrix

Posterior mean at t, given observations including t.
Steady-State Backwards Pass (Kalman Smoothing)

$$\bar{x}_t = J_{ss} \bar{x}_{t+1} + (I - J_{ss}A) \hat{x}_t$$

Doesn’t depend on observation dimension. Fast.
LDS for Text
Gaussian Likelihood for Words?

One-hot encoding

\[ [0, \ldots, 1, \ldots, 0] \]

“CAT”

Effect of using Gaussian Likelihood

<table>
<thead>
<tr>
<th>CAN DO</th>
<th>CAN NOT DO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perform Posterior Inference</td>
<td>Generate Text</td>
</tr>
<tr>
<td>Evaluate Probability of Observation</td>
<td></td>
</tr>
<tr>
<td>Fit Model Very Quickly</td>
<td></td>
</tr>
</tbody>
</table>
Relationship to RNN Language Model

\[ \hat{x}_t = (A - K_{ss}CA)\hat{x}_{t-1} + K_{ss}w_t \]

Product with one-hot vector = word embedding lookup

Kalman filter updates = RNN language model updates with no non-linearities
# Text-LDS vs. RNN Language Model

<table>
<thead>
<tr>
<th>Pros</th>
<th>Cons</th>
</tr>
</thead>
</table>
| **LDS** | • Fast learning (this paper)  
• Backwards Pass | • Can’t generate text from it.  
• Perplexity uninterpretable |
| **RNN-LM** | • longer-term memory | • slow training  
• difficult to tune stepsizes, etc. |

**Spoiler Alert:**

We speed up RNN training by initializing with LDS parameters.
Learning the LDS Parameters
Type-Level Sufficient Statistics

\[ \Psi_i = \mathbb{E}_t [w_{t+i} w_t^\top] \]

\[ [\Psi_i]_{jk} = \frac{\# \text{ (word}_k \text{ i positions to the right of word}_j \text{ )}}{N} \]

Collect in single (parallelizable) pass over corpus.

Spectral learning of HMMs uses third order moments

\[ \mathbb{E}_t [w_{t+2} \otimes w_{t+1} \otimes w_t] \text{ difficult to estimate!} \]
Learning Algorithm 1: Subspace Identification (Method of Moments)

(Van Overschee & De Moor, 1996)

**Step 1:** Construct Big, Sparse Hankel Matrix

\[ H_r = \begin{pmatrix} \Psi_r & \Psi_{r-1} & \Psi_{r-2} & \ldots & \Psi_1 \\ \Psi_{r+1} & \Psi_r & \Psi_{r-1} & \ldots & \Psi_2 \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \Psi_{2r-1} & \Psi_{2r-2} & \Psi_{r-3} & \ldots & \Psi_r \end{pmatrix} \]

**Step 2:** (Randomized) SVD (Halko and Tropp, 2009)

\[ H_r = \Gamma_r \Delta_r \]

<table>
<thead>
<tr>
<th>PROS</th>
<th>CONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fast, Non-Iterative</td>
<td>Statistically Suboptimal</td>
</tr>
<tr>
<td>Statistically Consistent</td>
<td></td>
</tr>
</tbody>
</table>
Two-Stage Estimation

Meta-Algorithm:
1) Initialize parameters
2) Do local search on likelihood surface using EM (because MLE is statistically optimal)
Learning Algorithm 2: Expectation-Maximization (Initialized With Subspace ID)

E-Step = Posterior inference over the corpus

M-Step = Two easy least-squares problems

Slow. Not at type-level.
ASOS E-Step (Martens, 2010)

**Observation 1:**
The M-step is least-squares, so all we need from the E step are time-averaged second order statistics.

\[
\mathbb{E}[\hat{x}_t w_t^\top], \quad \mathbb{E}[^\top \hat{x}_t \hat{x}_t], \quad \mathbb{E}[\hat{x}_{t+1}^\top \hat{x}_t]
\]

**Observation 2:**
If the posterior follows a Markov relationship (Kalman Filter), then so do the time-averaged second order statistics.

Example Markov relationship

\[
x_t = A x_{t-1} + b_t
\]

Markov relationship on second-order statistics

\[
\mathbb{E}[x_t w_t^\top] = A \mathbb{E}[x_{t-1} w_t^\top] + \mathbb{E}[b_t w_t^\top]
\]

**Observation 3:**
Using \(\Psi_i\), we can Kalman filter + smooth second-order statistics matrices directly!
Recap

So far: how to handle very large corpora.

Next: how to handle large vocabularies by exploiting the specific structure of one-hot data.
High Dimensional Observations

\[ x_t = A x_{t-1} + \eta \]
\[ w_t = C x_t + \epsilon, \]
\[ \epsilon \sim N(0, D), \eta \sim N(0, Q) \]

Can't even store a V x V matrix!

Option 1: Use diagonal approximation.

Option 2: Exploit specific functional form of MLE for D
MLE for Noise Covariance

\[ \mu = \text{vector of word frequencies} \]

\[ \Psi_0 = \mathbb{E}_t[w_tw_t^\top] = \text{diag}(\mu) - \mu\mu^\top \]

MLE noise covariance is diagonal-minus-low-rank:

\[ I - \mu \frac{1}{2} \mu \frac{1}{2}^\top + \left[ CM^\top \right] B \left[ E^\top M \right]^\top \]

But we need the \textit{inverse} covariance all over the place...

\textit{Sherman-Woodbury-Morrison to the rescue!}
More Linear Algebra Tricks (see paper)

• Whiten the data for SSID using unigram frequencies.

• Account for rank deficiency of the one-hot observations.
Obtaining Token Embeddings using the LDS
Train Time:
1. Train the LDS
2. Find posterior latent covariance on training data
3. Transform LDS so that training latent covariance is spherical

Test Time:
1. Run Kalman smoothing per-sentence to get posterior over latent states.
2. Token Embedding = Posterior Mean
Experiments
The transition matrix $A$ converts right singular vectors into left singular vectors. Are these interpretable?

<table>
<thead>
<tr>
<th>Right Singular Vector</th>
<th>Left Singular Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>chris mike steve jason tim jeff bobby ian greg adam tom phil nick brian ron</td>
<td>evans anderson harris robinson smith phillips collins murray murphy</td>
</tr>
<tr>
<td>brooklyn art science harlem princeton manhattan wimbledon hartford arts</td>
<td>symphony journal briefing street harbor beach birthday medal avenue bay innings</td>
</tr>
<tr>
<td>manhattan wimbledon hartford arts</td>
<td>box park district</td>
</tr>
<tr>
<td>greenwich advertising massachusetts</td>
<td></td>
</tr>
<tr>
<td>salt chicken pepper chocolate butter cheese cream sauce bread sugar thick</td>
<td>chicken cream pepper sauce cheese chocolate salt butter bread sweet</td>
</tr>
<tr>
<td>policemen helicopters soldiers suspects demonstrators guards iraqis personnel</td>
<td>remained expressed recommended denied remains feels gets resumed is sparked</td>
</tr>
</tbody>
</table>
WSJ Part of Speech Tagging

Method:
Local classification using dense features per token.

<table>
<thead>
<tr>
<th>Method</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>Word2Vec</td>
<td>92.58</td>
</tr>
<tr>
<td>LDS-SSID</td>
<td>83.00</td>
</tr>
<tr>
<td>LDS-EM</td>
<td>94.30</td>
</tr>
</tbody>
</table>

Remarks:
1. SSID performs poorly on its own.
2. The LDS sequence model outperforms Word2Vec.
WSJ Part of Speech Tagging

Method:
Structured prediction using dense + lexicalized features

<table>
<thead>
<tr>
<th>Method</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lex</td>
<td>97.28</td>
</tr>
<tr>
<td>Lex + LDS-EM</td>
<td>97.32</td>
</tr>
<tr>
<td>Lex + Word2Vec</td>
<td>97.35</td>
</tr>
</tbody>
</table>

Remarks:
LDS sequence modeling unnecessary when performing global structured prediction.
RNN Initialization

- The Kalman Filter updates are identical to the those of an RNN with no non-linearities.
- Non-linear RNN training with SGD is slow.
- Initialize the RNN with LDS parameters!
Conclusion

• We obtain context-dependent word embeddings by performing posterior inference in an LDS.
• You can learn continuous latent state sequence models using only type-level statistics!
• Our LDS is a simple, scalable alternative to an RNN. Usefulness:
  – Current work: initialize RNN with LDS parameters.
  – Future: use within variational latent-variable RNN frameworks.
• Code coming soon. Check my website.
Questions?
Learning Algorithm: Overview

Step 1: Gather $\Psi_i = \mathbb{E}_t[w_{t+i}w_t^\top]$  

Step 2: Estimate LDS parameters using Subspace Identification (Method of Moments)

Step 3: Perform about 50 iterations of EM to refine parameters.

Steps 2 and 3 only operate on $\Psi_i$
NER Tagging

Method:
Structured Prediction using
Dense + Lexicalized Features

<table>
<thead>
<tr>
<th></th>
<th>Lex</th>
<th>Lex + Brown</th>
<th>Lex + Word2Vec</th>
<th>Lex + LDS-EM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>89.3</td>
<td>89.8</td>
<td>90.0</td>
<td>89.9</td>
</tr>
</tbody>
</table>

Remarks:
Similar gain as established benchmarks.
Subspace ID (continued)

Step 2: SVD

\[ H_r = \Gamma_r \Delta_r \]

Step 3: Use Nested Structure to Recover \( A \) and \( C \) using Least Squares

\[ \Gamma_r = \begin{bmatrix} C \mid CA \mid CA^2 \mid \ldots \mid CA^{r-1} \end{bmatrix} \]

\[ \Delta_r = \begin{bmatrix} A^{r-1}G \mid A^{r-2}G \mid \ldots \mid AG \mid G \end{bmatrix} \]
LDS on Projected Words

• Step 1:
  Train type-level word embeddings using some existing algorithm.

• Step 2:
  Project the unsupervised training corpus.

• Step 3:
  Fit an LDS on the projected data.
LDS on Projected Words

• Advantages
  – Gaussian assumption is more reasonable.
  – Linear algebra tricks are unnecessary for scalability

• Problems
  – Still can’t generate text from it.
  – Vulnerable to choice of embeddings.
LDS on Projected Words

New random variable:

\[ Mw_t \]

Covariance of projection = projection of covariance:

\[
E_t[Mw_t(Mw_t)^	op] = ME_t[w_tw_t^	op]M^\top
\]
Motivation: EM vs. SGD

- Tuning learning rate schedules for non-convex problems is annoying and difficult.
- EM takes big batch steps on the likelihood.
<table>
<thead>
<tr>
<th></th>
<th>Word2Vec</th>
<th>LDS-SSID</th>
<th>LDS-EM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal</td>
<td>95.00</td>
<td>89.26</td>
<td>96.44</td>
</tr>
<tr>
<td>Penn</td>
<td>92.58</td>
<td>83.00</td>
<td>94.30</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Lex</th>
<th>Lex + LDS-EM</th>
<th>Lex + Word2Vec</th>
</tr>
</thead>
<tbody>
<tr>
<td>Universal</td>
<td>97.97</td>
<td>98.05</td>
<td>98.02</td>
</tr>
<tr>
<td>Penn</td>
<td>97.28</td>
<td>97.32</td>
<td>97.35</td>
</tr>
</tbody>
</table>
Neural Language Model
(Mnih and Hinton, 2007)

Represent context as linear combination of context words’ embeddings

\[ \hat{r} = \sum_{i=1}^{n-1} C_i r_{w_i} \]

Word probability is log-bilinear

\[ P(w_n = w | w_{1:n-1}) = \frac{\exp(\hat{r}^T r_w + b_w)}{\sum_j \exp(\hat{r}^T r_j + b_j)} \]
ASOS
(Martens, 2010)

**Step 0:**
Collect empirical covariances at various lags

\[ \Psi_i = E_t[w_{t+i}w_t^\top] \]

**Step 1:**
Approximate covariances at high lags by assuming that they are drawn from the current model parameters.

**Step 2:**
Run a Kalman filter on the second order statistics directly.

**Step 3:**
Use the estimated covariances at lag = 0 to perform the M step.
Learning Algorithm: Overview

Gather Sufficient Statistics

$$\Psi_i = \mathbb{E}_t[w_{t+i} w_t^\top]$$

Subspace Identification
Motivation: Using Co-Occurrence Counts

• Learning is *independent of corpus size*.
• Can apply type-level smoothing.
### Consideration: Sequence Model

Method Based on a Sequence Model?

<table>
<thead>
<tr>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brown Clusters</td>
<td>Word2Vec</td>
</tr>
<tr>
<td>Recurrent Neural Networks</td>
<td>Glove</td>
</tr>
<tr>
<td>POS Induction with HMMs</td>
<td>CCA</td>
</tr>
</tbody>
</table>