Differentiable Filtering for Learning Hidden Markov Models

Reginald Zhiyan Chen RZCHEN2@ILLINOIS.EDU

Heng-Sheng Chang* HSCHANG@ILLINOIS.EDU

Prashant G. Mehta MEHTAPG@ILLINOIS.EDU

University of Illinois Urbana-Champaign

Editors: G. Sukhatme, L. Lindemann, S. Tu, A. Wierman, N. Atanasov

Abstract

Hidden Markov Models (HMMs) are fundamental for modeling sequential data, yet learning their parameters from observations remains challenging. Classical methods like the Baum-Welch algorithm are computationally intensive and prone to local optima, while modern spectral algorithms offer provable guarantees but may produce probability outputs outside valid ranges. This work introduces Belief Net, a differentiable filtering framework that learns HMM parameters by formulating the forward filter as a structured neural network and optimizing it with stochastic gradient descent. This architecture recursively updates the belief state, which represents the posterior probability distribution over hidden states based on the observation history. Unlike black-box transformer models, Belief Net's learnable weights are explicitly the logits of the initial distribution, transition matrix, and emission matrix, ensuring full interpretability. The model processes observation sequences using a decoder-only (causal) architecture and is trained end-to-end with standard autoregressive next-observation prediction loss. On synthetic HMM data, Belief Net achieves faster convergence than Baum-Welch while successfully recovering parameters in both undercomplete and overcomplete settings, whereas spectral methods prove ineffective in the latter. Comparisons with transformer-based models are also presented on real-world language data.

Keywords: Hidden Markov Models, Sequence Modeling, Transformer

1. Introduction

Hidden Markov Models (HMMs) constitute a fundamental class of probabilistic models for discrete-time sequential data, with broad applications spanning speech recognition (Rabiner, 2002), natural language processing (Manning and Schutze, 1999), computational biology (Durbin et al., 1998), and financial time series analysis (Hassan and Nath, 2005). In an HMM, an observed sequence is generated (emitted) from an unobserved sequence of discrete latent states that evolve as a Markov chain. A time-homogeneous model is fully characterized by three sets of parameters: an initial state distribution, a state transition matrix for the Markov chain, and an emission matrix defining the conditional distribution of observations given latent states (Murphy, 2012; Elliott et al., 1995).

The learning problem, or the system identification problem, is to recover the model parameters of the HMM from the observed sequences. A classical approach is the Baum-Welch algorithm (Baum et al., 1970), which is a special case of the Expectation-Maximization (EM) algorithm. While widely used, EM is an iterative, non-convex optimization method that is sensitive to initialization and often converges to poor local optima (Wu, 1983). More recently, spectral algorithms have emerged as an efficient and provably correct alternative (Hsu et al., 2012; Boots et al., 2011;

* Corresponding author. Coordinated Science Laboratory. Department of Mechanical Science and Engineering.

Balle et al., 2014). These spectral approaches apply singular value decomposition (SVD) to empirical probabilities, the *moments* of observation singles, pairs, and triples, to identify an *observable representation*, yet they often fail in overcomplete regimes due to rank deficiencies and produce outputs that fall outside valid probability ranges (Balle and Maillard, 2017).

In parallel, the field of deep learning has produced powerful general-purpose sequence-to-sequence models, such as transformers (Vaswani et al., 2017), that excel at next-step prediction through gradient-based optimization (Bottou, 2010). These models have demonstrated remarkable capabilities in modeling sequential data through an attention mechanism, which captures long-range dependencies and complex patterns in sequences (Tay et al., 2020). Unlike an HMM, the parameters of transformer models are not readily interpretable (Rudin, 2019). While they do not explicitly recover the underlying generative structure of the model (Lipton, 2018), they have consistently achieved superior predictive performance on various tasks (Brown et al., 2020; Dosovitskiy, 2020).

Motivated by the relationship between the HMM learning problem and the modeling capabilities of transformer architectures, this paper explores two interrelated questions:

- How well does a transformer perform when the data is generated by an HMM?
- How well does an HMM-based learning algorithm perform on real-world language data where transformers excel?

To help answer these questions, we introduce a transformer-inspired gradient-based algorithm, referred to as the *Belief Net*, for learning an HMM. Closely mirroring the decoder-only architecture of modern auto-regressive language models, Belief Net is designed to perform one-step-ahead prediction by maintaining a *belief state* that encodes the observation history. This design choice enables end-to-end training with the same cross-entropy loss used in language modeling, while ensuring that the learned parameters remain interpretable as HMM transition and emission matrices. Our contributions are as follows:

- We formulate the HMM's recursive belief state update (the "forward filter") as a structured neural network whose learnable weights includes the logits of the initial distribution, transition matrix, and emission matrix.
- We show that this model can be trained end-to-end using backpropagation on the standard auto-regressive (next-observation prediction) cross entropy loss function, exactly like a modern decoder-only language model.
- On synthetic data generated by an HMM, we empirically demonstrate that Belief Net is faster than Baum-Welch and can recover parameters in settings where spectral algorithms fail.
- On real-world textual data, Belief Net learns an interpretable HMM that serves as a baseline predictive performance against a black-box transformer.

The remainder of this paper is organized as follows. Section 2 provides a brief overview of HMMs and reviews relevant learning algorithms. Section 3 introduces the proposed Belief Net framework, including its model architecture and gradient-based parameter optimization scheme. Section 4 presents the experimental evaluation, comparing Belief Net with both classical and modern baselines on synthetic benchmarks and a real-world language modeling task. Section 5 concludes the paper with a summary of key results and directions for future research.

2. Preliminaries and Related Work

2.1. Hidden Markov Models

A Hidden Markov Model characterizes a discrete-time stochastic process $\{(X_t, Z_t) \in \mathbb{S} \times \mathbb{O}\}_{t \geq 0}$, where $\mathbb{S} = \{\mathsf{x}_1, \ldots, \mathsf{x}_\mathsf{d}\}$ and $\mathbb{O} = \{\mathsf{z}_1, \ldots, \mathsf{z}_\mathsf{m}\}$ are the hidden (latent) state and observation spaces, respectively. For discrete-time steps $t \geq 0$, the latent state X_t evolves according to the Markov property, while the observation Z_t is generated conditionally on the current hidden state X_t. The model is fully characterized by the tuple (μ, A, C): The distribution of initial state X_0 is given by $\mu(\mathsf{x}) := \mathsf{P}(X_0 = \mathsf{x})$ for $\mathsf{x} \in \mathbb{S}$. The transition and emission probability matrices are $A_{ij} := \mathsf{P}(X_{t+1} = \mathsf{x}_j \,|\, X_t = \mathsf{x}_i)$ and $C_{ik} := \mathsf{P}(Z_t = \mathsf{z}_k \,|\, X_t = \mathsf{x}_i)$ for $\mathsf{x}_i, \mathsf{x}_j \in \mathbb{S}$, $\mathsf{z}_k \in \mathbb{O}$, and $t \geq 0$.

Learning problem Given a dataset of N observation sequences $\mathcal{D} = \{Z_{0:T}^{(n)}\}_{n=1}^{N}$ generated by the HMM, whose parameters are unknown to the learner, the goal is to estimate the HMM parameters (μ, A, C). The number of learnable parameters is $\mathsf{d} - 1$ for μ, $\mathsf{d}(\mathsf{d} - 1)$ for A, and $\mathsf{d}(\mathsf{m} - 1)$ for C, where (-1) is because of the normalization constraints on probabilities. After being learned *offline*, the HMM can then be applied to a range of inference tasks, including filtering, smoothing, and predicting future observations.

2.2. Methods for HMM Learning

Several methods have been proposed for learning HMM parameters from observation data:

Baum-Welch Algorithm The Baum-Welch algorithm is a classic Expectation-Maximization algorithm used to estimate HMM parameters (Baum et al., 1970). It iteratively performs an E-step, which uses the smoothing algorithm to compute the expected state distributions given the observations, followed by an M-step, which re-estimates the parameters (μ, A, C) by calculating the expected state and observation occupancies based on those distributions. The algorithm's objective is to maximize the log-likelihood, and while it is guaranteed to find a local maximum, its performance is sensitive to initialization and it may converge to a poor local optimum.

Spectral Algorithm A spectral based algorithm (Hsu et al., 2012) is employed to identify an observable representation via SVD of empirical probability matrices. This approach circumvents local optima and is computationally efficient, as it does not require iterative training. However, its theoretical guarantees rely on a rank condition of the underlying HMM parameters, and the outputs are not necessarily valid probability distributions. A simplified version is provided in Appendix B.2.

General Sequence Models (Transformers/RNNs) Models such as RNNs (Rumelhart et al., 1985), LSTMs (Hochreiter and Schmidhuber, 1997), and transformers (Vaswani et al., 2017) can be trained on the same next-observation prediction and are highly expressive; however, their parameters do not correspond to the underlying HMM parameters, rendering them uninterpretable black-box models. Related work on learning state-space models with differentiable filtering is reviewed in Appendix A.

A key motivation is to adapt the transformer-based loss function, input-output architecture, and training algorithms, now for learning parameters of an HMM. To help relate the two, the state dimension d is set equal to the embedding dimension of a transformer. This is primarily because the number of parameters, e.g., in the attention projection matrices and dense feed-forward networks, scales quadratically with the embedding dimension (Geva et al., 2021), analogous to the scaling of parameters in an HMM with respect to the state dimension.

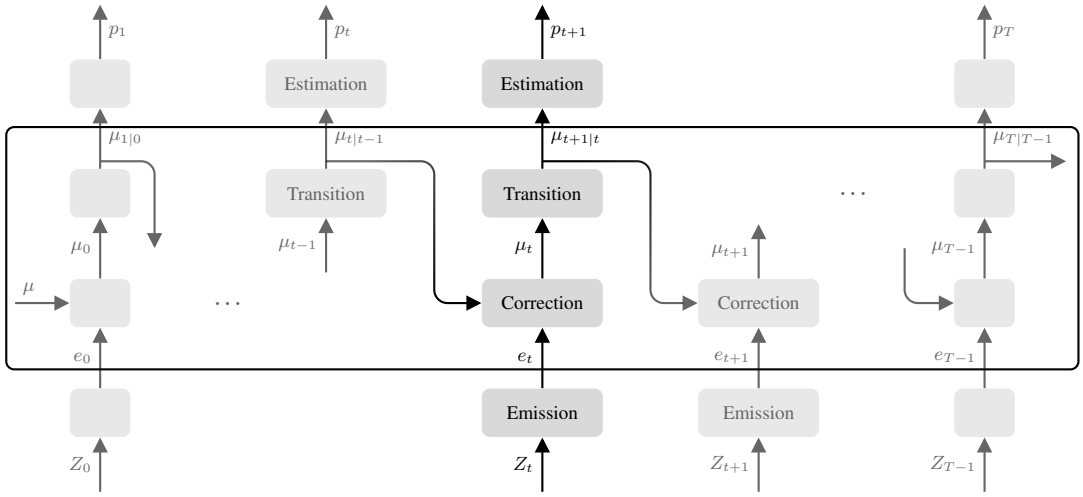

Figure 1: Belief Net architecture. The model initializes at $t = 0$ with μ as prior and maintains a belief state μ_t over the sequence $t \in \{0, 1, \ldots, T-1\}$ by recursively applying transition (using transition matrix A) and correction (using previous prior $\mu_{t|t-1}$ and e_t from emission step based on observation Z_t and emission matrix C) steps to update beliefs. The estimation step is to predict probabilities of the next observation p_{t+1} based on the next prior $\mu_{t+1|t}$ and emission matrix C. The detailed computation of each step is described in Algorithm 1.

This design allows for a direct comparison between transformers and Belief Net performance. Such a comparison is useful for two reasons:

- On synthetic data generated by an HMM, it reveals how well a transformer can learn the underlying structure when the data is truly generated by an HMM.
- On real-world language data, it quantifies the performance gain that a transformer achieves compared to an interpretable HMM-based algorithm, thereby providing insights into the non-Markovian nature of the embedded latent process in language models.

3. The Belief Net Framework

This paper proposes to learn the HMM parameters $\theta = (\mu, A, C)$ directly by formulating the HMM filter as a transformer-inspired neural network and using stochastic gradient descent to optimize parameters with respect to the standard next-observation prediction loss function, and then compute the estimation (the one-step prediction) $p_{t+1}(\mathsf{z}) := \mathsf{P}\left(Z_{t+1} = \mathsf{z} \,|\, Z_{0:t}\right)$ during inference. The core idea is to represent the recursive filtering equations as a computational graph (Scarselli et al., 2008; Hamilton, 2022), where the learnable weights correspond directly to the logits of the HMM parameters. A comparison with the Baum-Welch algorithm is also presented at the end of this section.

3.1. Model Architecture

Parameterization Belief Net is parameterized by learnable weights $\tilde{\theta} = (\tilde{\mu}, \tilde{A}, \tilde{C})$, where $\tilde{\mu} \in \mathbb{R}^{\mathsf{d}}$ is a row vector of logits for the initial state distribution; $\tilde{A} \in \mathbb{R}^{\mathsf{d} \times \mathsf{d}}$ is the logits for the transition matrix; and $\tilde{C} \in \mathbb{R}^{\mathsf{d} \times \mathsf{m}}$ is the logits for the emission matrix. The softmax operation is applied to obtain valid probability distributions, including

$$\mu = \text{softmax}\left(\tilde{\mu}\right), \quad A_{i,:} = \text{softmax}(\tilde{A}_{i,:}), \quad C_{i,:} = \text{softmax}(\tilde{C}_{i,:}), \quad \forall i \in \{1, 2, \ldots, \mathsf{d}\} \tag{1}$$

The complete set of probability parameters is denoted as $\theta = (\mu, A, C)$, which are functions of the logits $\tilde{\theta}$. Thus, for notational simplicity, the transformation from logits to probabilities is denoted as $\theta = \text{softmax}(\tilde{\theta})$.

Algorithm 1 Belief Net $f_{\tilde{\theta}}$

Parameters: parameters $\tilde{\theta} = (\tilde{\mu}, \tilde{A}, \tilde{C})$
Input: observation sequence $Z_{0:T-1}$,
Output: predicted observation distributions $p_{1:T}$

$(\mu, A, C) \leftarrow \text{softmax}(\tilde{\theta})$ `// to probability`
$\mu_{0|-1} \leftarrow \mu$ `// prior initialization`
for $t = 0$ **to** $T-1$ **do**
  `// Obtain observation` $Z_t = \mathsf{z}_k$
  `// Emission (likelihood)`
  $e_t \leftarrow C_{:,k}$
  `// Correction (posterior)`
  $\tilde{\mu}_t \leftarrow e_t \mu_{t|t-1}$
  $\mu_t \leftarrow \tilde{\mu}_t / \text{sum}(\tilde{\mu}_t)$
  `// Transition (next prior)`
  $\mu_{t+1|t} \leftarrow \mu_t A$
  `// Estimation (next observation)`
  $p_{t+1} \leftarrow \mu_{t+1|t} C$
end
return $p_{1:T}$

Algorithm 2 Belief Net Learning Process

Input: Dataset $\mathcal{D} = \{Z_{0:T}^{(n)}\}_{n=1}^{N}$ of observation sequences
Output: Estimated HMM parameters $\hat{\theta} = (\mu, A, C)$

$l \leftarrow 0$ `// iteration counter initialization`
Initialize learnable parameters $\tilde{\theta}^{(l)}$ randomly
Initialize optimizer AdamW with learning rate schedule η_l
while *not converged* **do**
  $\mathcal{D}_i \subset \mathcal{D}$ `// Sample a mini-batch`
  `/* Forward: use belief net to compute loss */`
  $\mathsf{J}_l \leftarrow \mathsf{J}(\tilde{\theta}^{(l)}; \mathcal{D}_l)$ `// equation (2)`
  `/* Backward: update parameters through backpropagation */`
  $\tilde{\theta}^{(l+1)} \leftarrow \tilde{\theta}^{(l)} - \eta_l \mathsf{AdamW}(\nabla_{\tilde{\theta}} \mathsf{J}_l)$
  `// Iteration counter increment`
  $l \leftarrow l + 1$
end
return $\hat{\theta} = \text{softmax}(\tilde{\theta}^{(l)})$

HMM Filter For an HMM, the *belief state*, $\mu_t(\mathsf{x}) := \mathsf{P}(X_t = \mathsf{x} \mid Z_{0:t})$ for $\mathsf{x} \in \mathbb{S}$, is the posterior distribution over the hidden state, given the history of observations $Z_{0:t}$, and it is a sufficient statistic for estimating the probability of next observation p_{t+1}. The posterior μ_t is computed recursively using the HMM filter (Elliott et al., 1995): for each step t, the prior, $\mu_{t|t-1}(\mathsf{x}) := \mathsf{P}(X_t = \mathsf{x} \mid Z_{0:t-1})$ for $\mathsf{x} \in \mathbb{S}$, is from the previous step $t-1$. It is the distribution over the hidden state X_t before observing the current observation Z_t. The likelihood is the probability of the current observation given the hidden state, computed through the emission process $e_t(\mathsf{x}) := \mathsf{P}(Z_t \mid X_t = \mathsf{x})$ for $\mathsf{x} \in \mathbb{S}$:

- *Emission Step*: $e_t(\mathsf{x}_i) = C_{i,k}$ observing $Z_t = \mathsf{z}_k$

The prior $\mu_{t|t-1}(\mathsf{x})$ for $\mathsf{x} \in \mathbb{S}$ is updated using the HMM filter, which consists of two steps:

- *Correction Step:* $\mu_t(\mathsf{x}) \propto e_t(\mathsf{x}) \mu_{t|t-1}(\mathsf{x})$
- *Transition Step:* $\mu_{t+1|t}(\mathsf{x}) = (\mu_t A)(\mathsf{x})$

The probability of the next observation $p_{t+1}(\mathsf{z})$ for $\mathsf{z} \in \mathbb{O}$ is then estimated as

- *Estimation Step:* $p_{t+1}(\mathsf{z}) = (\mu_{t+1|t} C)(\mathsf{z})$

This recursive update of the belief state μ_t forms the core of our proposed model, the *Belief Net*.

Belief Net Model Given an input sequence $Z_{0:T-1}$, the model $f_{\tilde{\theta}}$ maintains belief states μ_t internally and processes the input observations sequentially using the HMM filter. The model then outputs the predicted observation distribution p_{t+1} for each step $t \in [0, T-1]$. Therefore, the model is expressed as

$$f_{\tilde{\theta}} : \mathbb{O}^T \to (\mathcal{P}(\mathbb{O}))^T, \quad Z_{0:T-1} \mapsto f_{\tilde{\theta}}(Z_{0:T-1}) = p_{1:T}$$

where $\mathcal{P}(\mathbb{O})$ denotes the probability simplex over the observation space $\mathbb{O}$. The complete model is presented in Algorithm 1, and the architecture of the same is illustrated in Figure 1.

3.2. Learning Process

The Belief Net model $f_{\tilde{\theta}}$ is trained by minimizing the average cross entropy over all sequences in each mini-batch $\mathcal{D}_l \subset \mathcal{D}$, a random subset of the full dataset, for each iteration l. This is identical to the training objective for a decoder-only language model:

$$\ell(p_{1:T}, Z_{1:T}) := -\frac{1}{T}\sum_{t=1}^{T} \log p_t(Z_t) \tag{2a}$$

$$\mathsf{J}(\tilde{\theta}; \mathcal{D}_l) := \frac{1}{|\mathcal{D}_l|}\sum_{n\in\mathcal{D}_l} \ell(f_{\tilde{\theta}}(Z^{(n)}_{0:T-1}), Z^{(n)}_{1:T}) \tag{2b}$$

The optimization problem is to find the optimal logits $\tilde{\theta}^*$ that minimize the expected loss over the entire dataset:

$$\tilde{\theta}^* = \arg\min_{\tilde{\theta}} \mathsf{E}_{\mathcal{D}_l\sim\mathcal{D}}\left(\mathsf{J}(\tilde{\theta}; \mathcal{D}_l)\right) \tag{3}$$

This is optimized directly using stochastic gradient descent. The estimated model parameters are recovered from the learned logits using the softmax transformation $\hat{\theta} = \text{softmax}(\tilde{\theta}^*)$ as in equation (1). The complete learning framework is presented in Algorithm 2.

This framework is analogous to the Baum-Welch algorithm, but instead of alternating between E-step (computing expectations) and M-step (maximizing log likelihood), a gradient-based updates is performed on all parameters simultaneously using modern automatic differentiation. The detailed comparison between the two algorithms is provided in the next subsection.

3.3. Comparison with Baum-Welch Algorithm

Belief Net and the Baum-Welch algorithm are both iterative approaches for estimating HMM parameters θ from observed sequences $Z_{0:T}$. Although they aim to optimize the same objective function, their computation and update mechanisms differ from each other. The underlying optimization strategies and computational complexity of the two are discussed in this subsection.

Objective Function Both algorithms optimize the same objective: the log-likelihood of the observed data with respect to the HMM parameters θ. By the chain rule of probability, the log-likelihood is decomposed as

$$\log \mathsf{P}\left(Z_{0:T} \,|\, \theta\right) = \sum_{t=0}^{T} \log \mathsf{P}\left(Z_t \,|\, Z_{0:t-1}, \theta\right)$$

where $Z_{0:-1}$ is defined as the empty sequence. This shows that maximizing the joint log-likelihood is equivalent to maximizing the sum of one-step-ahead conditional log-likelihoods. Both Baum-Welch and Belief Net aim to find parameters θ that optimize this objective over the dataset $\mathcal{D}$:

$$\max_{\theta} \mathsf{E}_{Z_{0:T}\sim\mathcal{D}}\left(\log \mathsf{P}\left(Z_{0:T} \,|\, \theta\right)\right)$$

which is equivalent to minimizing the negative log-likelihood of the observed data with respect to the HMM parameters θ as in equation (3). The key difference between the two algorithms lies not in the objective function itself, but in how the algorithms optimize it.

Update Mechanism The main difference is the strategies for optimizing the same objective.

- *Baum-Welch (EM):* The Baum-Welch algorithm, based on the Expectation-Maximization framework, iteratively maximizes a lower bound $\mathsf{Q}(\theta;\theta^{(l)})$ on the log-likelihood.

$$\log \mathsf{P}\left(Z_{0:T}|\theta\right) \geq \mathsf{Q}(\theta;\theta^{(l)}) := \mathsf{E}_{\mathsf{P}\left(X_{0:T}\,|\,Z_{0:T},\theta^{(l)}\right)}\left(\log \mathsf{P}\left(X_{0:T}, Z_{0:T}\,|\,\theta\right)\right)$$

  It computes expected sufficient statistics through smoothing and updates parameters in closed form through the necessary condition for maximization:

$$\theta^{(l+1)} = \arg\max_{\theta} \mathsf{Q}(\theta;\theta^{(l)})$$

- *Belief Net (Gradient-Based):* Parameters are updated through the optimizer AdamW (Loshchilov and Hutter, 2017) with automatic differentiation on the objective equation (2).

$$\tilde{\theta}^{(l+1)} = \tilde{\theta}^{(l)} - \eta_l \mathsf{AdamW}(\nabla_{\tilde{\theta}} \mathsf{J}(\tilde{\theta}^{(l)};\mathcal{D}_l)), \quad l = 0, 1, \ldots$$

  Here, at each iteration l, $\eta_l > 0$ is the learning rate, $\mathsf{J}(\tilde{\theta}^{(l)};\mathcal{D}_l)$ is the log-likelihood over the mini-batch $\mathcal{D}_l$, and $\tilde{\theta}^{(l)}$ are the logits corresponding to the HMM parameters.

Remark 1 *The key properties of the expectation-maximization algorithm, including monotonic improvement of the log-likelihood and consistency of convergence to a stationary point, are well-established in the literature (Dempster et al., 1977; Wu, 1983; Krishnamurthy, 2016).*

Remark 2 *While gradient-based optimization methods, such as stochastic gradient descent and its variants (e.g., AdamW), do not enjoy the same guarantees on monotonic improvement and consistency of convergence as in the EM algorithm, they have been known for their scalability and flexibility in handling large datasets and complex models (Bottou et al., 2018).*

Remark 3 *Prior work has applied gradient-based methods to learn HMMs without employing the filtering architecture (Bagos et al., 2004). More recent neural-HMM hybrids further extend this idea (Rimella and Whiteley, 2025). While sharing the gradient-based learning philosophy, these approaches differ from Belief Net in objectives, architecture, and interpretability. More detailed discussion on this neural-HMM hybrid line of work is provided in Appendix A.*

Computational Complexity The computational complexity of the two algorithms differs on two fronts: the amount of data processed per iteration and the operations required per sequence.

- *Baum-Welch* processes the entire dataset of N sequences in each iteration, requiring full smoothing paths for all sequences.
  - Data processed per iteration: N sequences of length T.
  - Operations per sequence: smoothing paths, expected state occupancy, and expected state transition all require $O\left(T\mathsf{d}^2\right)$ operations per sequence.
  - Parameter updates are computed in closed-form with $O\left(\mathsf{d}^2 + \mathsf{dm}\right)$ operations.

  The total complexity per iteration is thus $O\left(NT\mathsf{d}^2\right)$.

- *Belief Net* processes a mini-batch of B sequences each iteration, allowing stochastic updates.
  - Data processed per iteration: B sequences of length T, where typically $B \ll N$.
  - Operations per sequence: Both the filtering path and backpropagation require the same order of operations $O\left(T(\mathsf{d}^2 + \mathsf{dm})\right)$ per sequence.
  - Parameter updates via gradient descent require $O\left(\mathsf{d}^2 + \mathsf{dm}\right)$ operations.

  The total complexity per iteration is thus $O\left(BT(\mathsf{d}^2 + \mathsf{dm})\right)$.

Since typically $B \ll N$, Belief Net processes substantially fewer sequences per iteration, enabling faster iteration times and better scalability.

4. Experiments

In this paper, the Belief Net framework is evaluated on two tasks:

- *Synthetic HMM Data:* The objective is to evaluate prediction accuracy and parameter recovery on synthetic data generated from HMMs.
- *Real-World Text Data:* The objective is to evaluate prediction performance on text data.

For both tasks, Belief Net is benchmarked against the Baum-Welch algorithm (from `hmmlearn` library), an independently implemented spectral algorithm, and two transformer-based models: a single-head single-layer model (nanoGPT-s) and a multi-head multi-layer model (nanoGPT-m), both trained from scratch.

The core objectives are to assess Belief Net's performance compared against both the classical HMM methods and also the transformer architectures, particularly when all models are matched for embedding dimension d. Studies with varying d are also presented. Detailed implementation and training setup for all methods are available in Appendix B.

4.1. Synthetic HMM Data: Prediction Accuracy and Parameter Recovery

For the synthetic data, the experimental setting are as follows:

Data Generation: Each dataset $\mathcal{D} = \{Z_{0:T}^{(n)}\}_{n=1}^{N}$ consists of N sequences of length $T = 256$. Observation sequences are generated from HMMs with parameters (μ, A, C) configured as follows:

- *Initial Distribution μ:* A uniform distribution over the hidden states.
- *Transition Matrix A:* A convex combination of a cyclic permutation matrix A^{cyclic} ($\mathsf{x}_1 \mapsto \mathsf{x}_2 \mapsto \cdots \mapsto \mathsf{x}_\mathsf{d} \mapsto \mathsf{x}_1$) and a random stochastic matrix A^{random}: $A = \alpha A^{\text{cyclic}} + (1-\alpha)A^{\text{random}}$, where the homotopy parameter $\alpha = 0.9$ controls the strength of the cyclic structure. This structure encourages state persistence while allowing for some randomness in transitions.
- *Emission Matrix C:* A random stochastic matrix.

Both stochastic matrices (A^{random} and C) are generated by sampling each entry of the matrix from a normal distribution, then normalizing each row using a softmax function with temperature. The temperatures are set to 0.1 for A^{random} and 0.01 for C to control the sparsity of the distributions. The validation dataset is generated from the same HMM parameters, but with a smaller number of sequences (10% of the training set size) to ensure sufficient validation data for model selection.

validation loss	Baum-Welch	Spectral	nanoGPT-s	nanoGPT-m	**Belief Net**	HMM Filter	Random
undercomplete	1.951	1.624	1.458	1.475	**1.569**	1.368	4.852
overcomplete	1.216	✗	0.829	0.810	**0.830**	0.737	3.466

# parameters	Baum-Welch	Spectral	nanoGPT-s	nanoGPT-m	**Belief Net**
undercomplete	12,352	524,416	73,920	221,760	**12,352**
overcomplete	6,208	✗	51,392	199,232	**6,208**

training time # iterations	Baum-Welch 20 iter.	Spectral (PCA)	nanoGPT-s 2,000 iter.	nanoGPT-m 2,000 iter.	**Belief Net** **2,000 iter.**
undercomplete	51 min	N/A	2.5 min	7.5 min	**20 min**
overcomplete	12 min	✗	1 min	1.5 min	**6 min**

- reference loss:
  - optimal: HMM Filter (model known)
  - random: $\ln(\mathsf{m})$
- ✗: fail (rank deficiency)
- N/A: PCA is instant

Table 1: Validation loss, number of parameters, and training time on synthetic data for each method.

4.1.1. PREDICTION ACCURACY

In this setting, the learner knows both the hidden state dimension d and the observation dimension m. Two cases are considered:

- *Undercomplete case (*$\mathsf{d} < \mathsf{m}$*):* Hidden states are fewer than observations (e.g., $\mathsf{d} = 64$, $\mathsf{m} = 128$). This is the typical scenario where the observation space is richer than the latent state space to which spectral methods are applicable. Number of samples in this case is $N = 4,000$.
- *Overcomplete case (*$\mathsf{d} > \mathsf{m}$*):* Hidden states are more than observations (e.g., $\mathsf{d} = 64$, $\mathsf{m} = 32$). This scenario tests whether methods can identify a higher-dimensional latent structure from a limited observation space (Sharan et al., 2017). Number of samples in this case is $N = 1,000$.

Each method's prediction accuracy (cross-entropy loss on a validation set) and training convergence speed (wall-clock time)[1] were recorded with results summarized in Table 1. The HMM Filter with true parameters is used to establish an optimal loss baseline; and random guessing is used as a worst-case baseline. The Belief Net framework consistently outperformed Baum-Welch algorithm, achieving low loss and faster convergence in both under- and overcomplete settings. The spectral method was only effective in the undercomplete case, failing in the overcomplete scenario due to rank deficiencies. The nanoGPT models achieved the lowest loss and successfully captured Markovian dependencies, consistent with Hu et al. (2024). However, contrary to their findings, the multi-head multi-layer model (nanoGPT-m) performs similarly to the single-head single-layer model (nanoGPT-s), suggesting that a simple architecture suffices to capture the latent Markovian structure. Additional results on emperical experiments on sensitivity analyses to initialization are provided in Appendix C.1.1.

4.1.2. PARAMETER RECOVERY

In this realistic setting, the learner only knows the observation dimension m. The dataset $\mathcal{D}$ is generated from an HMM with $\mathsf{d} = 64$ and $\mathsf{m} = 128$. To evaluate robustness to model misspecification, each model is trained and validated across a range of candidate state dimensions $\hat{\mathsf{d}} \in \{4, 8, 16, 32, 64, 128, 256\}$. As shown in Figure 2, validation loss is minimized when the

1. All training computations were conducted on a laptop with an Intel Core i7-12700H processor (2.3GHz, 6P+8E cores), 32GB 4800MHz DDR5 memory, and no GPU acceleration.

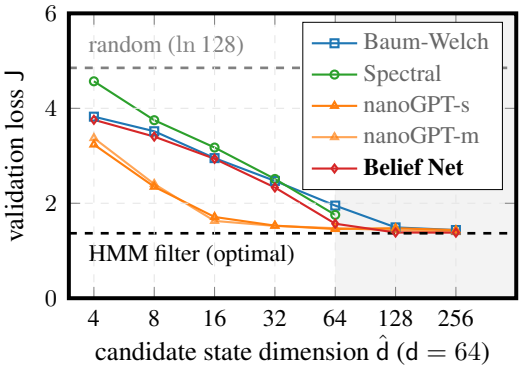

Figure 2: Parameter recovery on synthetic data. The validation loss J is plotted with respect to candidate state dimensions $\hat{d}$ for each method. The true state dimension is $d = 64$ and the gray area indicates the $\hat{d} \geq d$ regime. Curves correspond to models (colors): Baum-Welch (blue), Spectral (green), two nanoGPTs (oranges), and Belief Net (red). Dashed lines: random guess (gray) and HMM filter (black) represents worst and best scenarios, respectively.

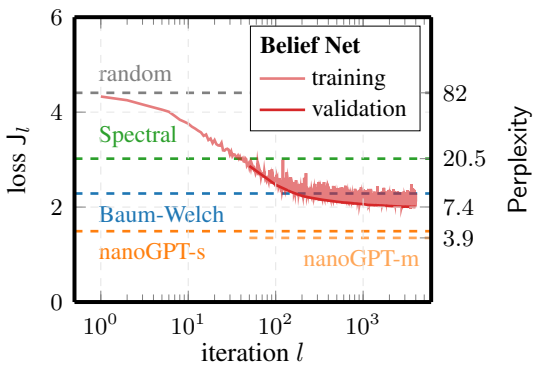

Figure 3: Language modeling results on Federalist Papers. The Belief Net's training (light red) and validation (red) loss J_l over iterations l are shown in solid curves. The validation loss is evaluated every fifty iterations. For comparison, horizontal dashed lines show the final validation losses achieved by other methods, including random guess (gray), Baum-Welch (blue), Spectral (green), and two nanoGPTs (oranges). Corresponding Perplexity is shown on the right.

candidate state dimension $\hat{d}$ matches or exceeds the true d, confirming its effectiveness for model selection. Across all settings, Belief Net outperforms Baum-Welch and spectral methods, approaching the optimal HMM filter when $\hat{d} \geq d$. Additional analyses of eigenvalue spectra and emission matrix discrepancies are provided in Appendix C.1.2. The nanoGPT models achieve the lowest and nearly identical losses, suggesting that a single-head, single-layer architecture suffices to capture the Markovian structure. Notably, nanoGPT-s at $\hat{d} = 16$ (9,264 parameters) and Belief Net at $\hat{d} = 64$ (12,352 parameters) attain comparable performance. Additional comparisons of Belief Net with nanoGPT models on various HMM settings are provided in Appendix C.1.2.

4.2. Real-World Text Data: Character-Level Language Modeling

Belief Net is evaluated on a language modeling task for next-token prediction using the Federalist Papers dataset (Jeong and Ročková, 2025; Bhatia, 2023). After tokenization, the dataset comprises $m = 82$ unique characters, with $N = 4,000$ training sequences; each sequence has a length of $T = 256$. The latent state dimension is fixed at $d = 64$ across all methods. Figure 3 depicts the training and validation losses, together with validation Perplexity $:= e^{J}$. The nanoGPT-m achieves the lowest perplexity, as expected given its capacity to model non-Markovian language structure. Belief Net outperforms classical methods, indicating improved modeling of Markovian latent dynamics, while retaining interpretability: its learned emission matrix captures the dominance of lowercase letters and identifies distinct states for uppercase and digit emissions, demonstrating recovery of meaningful latent structure. Further details are provided in Appendix C.2.

5. Conclusion

This paper introduces Belief Net, a differentiable filtering framework that bridges system identification and neural representation learning. Experiments show that it outperforms Baum-Welch in convergence speed, recovering parameters even in overcomplete regimes where spectral methods fail. On real text data, Belief Net achieves better performance compared to classical methods. Future work will extend the framework to more general settings, including POMDPs, non-Markovian models with memory beyond a single time step, and online learning, thereby broadening its applicability to a wider class of sequential inference and decision-making problems.

Acknowledgments

This work is supported in part by the AFOSR award FA9550-23-1-0060 and the NSF award 2336137. We gratefully acknowledge Dr. Tixian Wang for his valuable insights into nanoGPT models and for his assistance with the initial numerical experiments.

References

Pantelis G Bagos, Theodore D Liakopoulos, and Stavros J Hamodrakas. Faster gradient descent training of hidden markov models, using individual learning rate adaptation. In *International Colloquium on Grammatical Inference*, pages 40–52. Springer, 2004.

Borja Balle and Odalric-Ambrym Maillard. Spectral learning from a single trajectory under finite-state policies. In *International Conference on Machine Learning*, pages 361–370. PMLR, 2017.

Borja Balle, Xavier Carreras, Franco M Luque, and Ariadna Quattoni. Spectral learning of weighted automata: A forward-backward perspective. *Machine learning*, 96(1):33–63, 2014.

Leonard E Baum, Ted Petrie, George Soules, and Norman Weiss. A maximization technique occurring in the statistical analysis of probabilistic functions of markov chains. *The annals of mathematical statistics*, 41(1):164–171, 1970.

Aatish Bhatia. Watch an a.i. learn to write by reading nothing but federalist papers. *The New York Times*, 2023.

Byron Boots, Sajid M Siddiqi, and Geoffrey J Gordon. Closing the learning-planning loop with predictive state representations. *The International Journal of Robotics Research*, 30(7):954–966, 2011.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational Statistics*, pages 177–186. Springer, 2010.

Léon Bottou, Frank E. Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *SIAM Review*, 60(2):223–311, 2018. doi: 10.1137/16M1080173.

John-Joseph Brady, Yuhui Luo, Wenwu Wang, Víctor Elvira, and Yunpeng Li. Regime learning for differentiable particle filters. In *2024 27th International Conference on Information Fusion (FUSION)*, pages 1–6. IEEE, 2024.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Heng-Sheng Chang. a python library for signal and system simulation, design, and analysis. `https://github.com/hanson-hschang/Signal-System`, 2025.

Reginald Chen and Heng-Sheng Chang. Repository of the belief net for hmm learning. `https://github.com/Rockostoneo/BeliefNet-HMM`, 2026.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society: series B (methodological)*, 39(1): 1–22, 1977.

Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

Richard Durbin, Sean R Eddy, Anders Krogh, and Graeme Mitchison. *Biological sequence analysis: probabilistic models of proteins and nucleic acids*. Cambridge university press, 1998.

Robert J Elliott, John B Moore, and Lakhdar Aggoun. *Hidden Markov models: estimation and control*. Springer, 1995.

Rémi Flamary, Nicolas Courty, Alexandre Gramfort, Mokhtar Z Alaya, Aurélie Boisbunon, Stanislas Chambon, Laetitia Chapel, Adrien Corenflos, Kilian Fatras, Nemo Fournier, et al. Pot: Python optimal transport. *Journal of Machine Learning Research*, 22(78):1–8, 2021.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 5484–5495, 2021.

Saeed Ghadimi and Guanghui Lan. Stochastic first-and zeroth-order methods for nonconvex stochastic programming. *SIAM journal on optimization*, 23(4):2341–2368, 2013.

William L Hamilton. The graph neural network model. In *Graph representation learning*, pages 51–70. Springer, 2022.

Md Rafiul Hassan and Baikunth Nath. Stock market forecasting using hidden markov model: a new approach. In *5th international conference on intelligent systems design and applications (ISDA'05)*, pages 192–196. IEEE, 2005.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Daniel Hsu, Sham M Kakade, and Tong Zhang. A spectral algorithm for learning hidden markov models. *Journal of Computer and System Sciences*, 78(5):1460–1480, 2012.

Jiachen Hu, Qinghua Liu, and Chi Jin. On limitation of transformer for learning hmms. *arXiv preprint arXiv:2406.04089*, 2024.

So Won Jeong and Veronika Ročková. From small to large language models: Revisiting the federalist papers. *arXiv preprint arXiv:2503.01869*, 2025.

Andrej Karpathy. `nanogpt`: The simplest, fastest repository for training/finetuning medium-sized gpts. `https://github.com/karpathy/nanoGPT`, 2024.

Alina Kloss, Georg Martius, and Jeannette Bohg. How to train your differentiable filter. *Autonomous Robots*, 45(4):561–578, 2021.

Vikram Krishnamurthy. *Partially observed Markov decision processes*. Cambridge university press, 2016.

Antony Lee and Matthew Danielson. `hmmlearn`: Hidden markov models in python, with scikit-learn like api. `https://github.com/hmmlearn/hmmlearn`, 2024.

Zachary C Lipton. The mythos of model interpretability: In machine learning, the concept of interpretability is both important and slippery. *Queue*, 16(3):31–57, 2018.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Christopher Manning and Hinrich Schutze. *Foundations of statistical natural language processing*. MIT press, 1999.

Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 2002.

Guy Revach, Nir Shlezinger, Xiaoyong Ni, Adria Lopez Escoriza, Ruud JG Van Sloun, and Yonina C Eldar. Kalmannet: Neural network aided kalman filtering for partially known dynamics. *IEEE Transactions on Signal Processing*, 70:1532–1547, 2022.

Lorenzo Rimella and Nick Whiteley. Hidden markov neural networks. *Entropy*, 27(2):168, 2025.

Cynthia Rudin. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence*, 1(5):206–215, 2019.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, 1985.

Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. The graph neural network model. *IEEE transactions on neural networks*, 20(1):61–80, 2008.

Vatsal Sharan, Sham M Kakade, Percy S Liang, and Gregory Valiant. Learning overcomplete hmms. *Advances in Neural Information Processing Systems*, 30, 2017.

Richard Sinkhorn. Diagonal equivalence to matrices with prescribed row and column sums. *The American Mathematical Monthly*, 74(4):402–405, 1967.

Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long range arena: A benchmark for efficient transformers. *arXiv preprint arXiv:2011.04006*, 2020.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

C. F. Jeff Wu. On the convergence properties of the em algorithm. *The Annals of Statistics*, 11(1): 95–103, 1983. ISSN 00905364, 21688966.

Yuan Wu and Sicheng He. Dkfnet: Differentiable kalman filter for field inversion and machine learning. *arXiv preprint arXiv:2509.07474*, 2025.

Han Zhao. Spectral-learning: The spectral learning procedure/oom model learning. `https://github.com/hanzhaoml/Spectral-learning`, 2014.

Appendix

The appendix provides additional details on the related work, implementation details, and additional experimental results.

- *Related Work:* A review of related literature on learning state-space models and connections to neural network architectures.
- *Implementation:* Details on the implementation of each model, including library usage, training procedure, and hyperparameters settings.
- *Experiments:* Additional experiment results complementing those in the main text, including training curves, initialization sensitivity analyses, evaluations of parameter recovery, additional comparisons with Belief Net and nanoGPTs, and interpretations of the learned models.

Appendix A. Related Work on Learning State-Space Models

This work is closely related to a broad class of architectures that learn state-space models by unrolling recursive updates into computational graphs, thereby enabling end-to-end training with gradient-based optimization (Scarselli et al., 2008; Hamilton, 2022). Such approaches have been particularly successful for continuous-state estimators, including neural variants of Kalman filters (Revach et al., 2022), and more recently within the broader paradigm of differentiable filtering (Kloss et al., 2021; Wu and He, 2025). Extensions to nonlinear and non-Gaussian systems have also been developed through differentiable particle filters, which integrate sequential Monte Carlo methods into deep learning frameworks for state inference (Brady et al., 2024). In contrast to these continuous-state formulations, where the belief state is typically represented as a mean-covariance pair or a set of weighted particles, Belief Net operates in a discrete setting, in which the belief state is a probability vector over discrete latent states.

A closely related line of work is Hidden Markov Neural Networks (HMNNs) (Rimella and Whiteley, 2025), which similarly combine HMM structure with neural architectures. However, the objectives differ fundamentally. HMNNs treat neural network weights themselves as latent HMM states, enabling continual learning as new data arrive. In contrast, Belief Net leverages a neural computation graph purely for offline system identification: its goals are (i) to recover the classical HMM parameters from historical observations, and (ii) to perform inference using the standard HMM filtering procedure with the learned parameters. In this sense, HMNNs extend what the latent states represent, whereas Belief Net focuses on how HMM parameters are estimated.

Appendix B. Implementation Details

This section provides detailed implementation specifications for all methods evaluated in Section 4. Section B.1 outlines the Baum-Welch algorithm implemented using the `hmmlearn` library. Section B.2 describes the spectral algorithm adapted from the `spectral-learning` repository, extended with a custom probability prediction function. Section B.3 details the transformer-based models implemented using the `nanoGPT` repository, which offers a minimal implementation of the transformer architecture. Finally, Section B.4 presents the Belief Net implementation using `PyTorch` for automatic differentiation and neural network computations.

B.1. Baum-Welch Algorithm

The `hmmlearn` Python library (Lee and Danielson, 2024) was utilized for all experiments involving the Baum-Welch algorithm. For each run, a `CategoricalHMM` object was configured with randomly initialized parameters, including the initial distribution μ, transition matrix A, and emission matrix C, and a maximum of 20 iterations was specified. The model was then trained on the dataset $\mathcal{D}$ using the `fit` method, which iteratively estimates the HMM parameters $\theta = (\mu, A, C)$. Upon reaching the iteration limit, the learned parameters $\hat{\theta} = (\hat{\mu}, \hat{A}, \hat{C})$ were extracted and used to evaluate the validation loss. To reduce sensitivity to random initialization, the procedure was repeated 5 times with distinct random seeds, and the run with the lowest validation loss was selected.

B.2. Spectral Algorithm

The spectral algorithm implementation follows the method described in Hsu et al. (2012). The model initialization and parameter estimation were adapted from `spectral-learning` (Zhao, 2014), while the probability prediction was based on the original paper (Hsu et al., 2012).

Empirical Probabilities Given training data $\mathcal{D}$, construct the empirical probability estimates $P_1, P_{2,1}, P_{3,k,1}$ by counting number of occurrences of overlapping subsequences of length three in the training data. For any observation $\mathsf{z}, \mathsf{z}' \in \mathbb{O}$ and for any t within the sequence, the following time-invariant empirical probability estimates are computed:

- Probability vector of dimension m: $P_1(\mathsf{z}) = \mathsf{P}\left(Z_t = \mathsf{z} \,|\, \mathcal{D}\right)$
- Probability matrix of dimension $\mathsf{m} \times \mathsf{m}$: $P_{2,1}(\mathsf{z}', \mathsf{z}) = \mathsf{P}\left(Z_{t+1} = \mathsf{z}', Z_t = \mathsf{z} \,|\, \mathcal{D}\right)$
- Matrix-valued probability function of dimension $\mathsf{m} \times \mathsf{m}$:

$$\mathbb{O} \to \mathcal{P}\left(\mathbb{O}^2\right), \quad \mathsf{z}_k \mapsto P_{3,k,1}(\mathsf{z}', \mathsf{z}) \propto \mathsf{P}\left(Z_{t+2} = \mathsf{z}', Z_{t+1} = \mathsf{z}_k, Z_t = \mathsf{z} \,\middle|\, \mathcal{D}\right)$$

Observable Representation To compute the observable representation, the SVD of $P_{2,1}$ is first performed: $P_{2,1} = \mathrm{U}\Sigma\mathrm{V}^\intercal$, where $\mathrm{U} \in \mathbb{R}^{\mathsf{m}\times\mathsf{m}}$ and $\mathrm{V} \in \mathbb{R}^{\mathsf{m}\times\mathsf{m}}$ are orthogonal matrices, and $\Sigma = \mathrm{diag}\{\sigma_1, \sigma_2, \ldots, \sigma_{\mathsf{m}}\}$ is a diagonal matrix of singular values σ_k sorted in descending order. The top d singular values $\sigma_{1:\mathsf{d}}$ are retained, and the corresponding left singular vectors are extracted to form the matrix $\mathrm{U}_\mathsf{d} \in \mathbb{R}^{\mathsf{m}\times\mathsf{d}}$. The observable representation parameters are computed as follows:

$$b_0 = \mathrm{U}_\mathsf{d}^\intercal P_1, \quad b_\infty = (P_{2,1}^\intercal \mathrm{U}_\mathsf{d})^+ P_1, \quad \mathrm{B}_k = \mathrm{U}_\mathsf{d}^\intercal P_{3,k,1} (\mathrm{U}_\mathsf{d}^\intercal P_{2,1})^+, \quad \forall \mathsf{z}_k \in \mathbb{O}$$

where the superscript $+$ denotes the Moore-Penrose pseudoinverse.

Recursive Update Once obtaining a new observation Z_t, the internal state b_t is updated following:

$$b_{t+1} = \frac{\mathrm{B}_k b_t}{b_\infty^\intercal \mathrm{B}_k b_t}, \quad \text{with } Z_t = \mathsf{z}_k$$

where the case of denominator being zero is handled by resetting $b_{t+1} = b_0$.

Prediction The conditional probability for the next observation Z_{t+1} given history $Z_{0:t}$ is:

$$\mathsf{P}\left(Z_{t+1} = \mathsf{z}_k \,|\, Z_{0:t}\right) \propto b_\infty^\intercal \mathrm{B}_k b_t, \quad \forall \mathsf{z}_k \in \mathbb{O}$$

To handle the cases of negative probabilities, all negative value entries were replaced with the minimum positive value at the current step, and the resulting vector was renormalized to sum to unity.

B.3. transformer Model

The transformer baselines were implemented using the `nanoGPT` repository (Karpathy, 2024), which provides a minimal decoder-only GPT architecture. Two configurations were evaluated to assess the impact of model capacity on performance:

- nanoGPT-s: A single-head, single-layer transformer represents the simplest architecture.
- nanoGPT-m: A multi-head, multi-layer transformer (4 heads, 4 layers) represents a more expressive variant.

Both models were configured with an embedding dimension of d, a feed-forward dimension of $4\mathsf{d}$, and learnable positional embeddings. Training was performed on a next-observation prediction task using cross-entropy loss. The hyperparameters included a batch size of 10, dropout values in $\{0.0, 0.1\}$, learning rates in $\{0.001, 0.01\}$, and a maximum of 2,000 iterations for synthetic data and 4,000 iterations for text data. A grid search over the dropout and learning rate values was conducted, with the configuration achieving the lowest validation loss selected for final evaluation. All remaining settings followed the repository defaults for CPU-only training. To support the experimental setup, the `train.py` script was modified to allow direct data loading from generated sequences, while the core model architecture defined in `model.py` remained unchanged.

B.4. Belief Net

The repositories containing the Belief Net implementation and the experiments presented in Section 4 are available in (Chang, 2025; Chen and Chang, 2026) and were developed as part of this work. The implementation follows Algorithms 1 and 2 in the main text. The model is trained using a batch size of 10, with dropout and learning rate selected from $\{0.0, 0.1\}$ and $\{0.01, 0.1\}$, respectively, via grid search based on validation loss. Training is run for up to 2,000 iterations on synthetic data and 4,000 iterations on text data. Optimization is performed using AdamW with betas $(0.9, 0.999)$, eps 10^{-8}, and weight decay 0.01, without AMSGrad, and with decoupled weight decay enabled.

Remark 4 *A theoretical analysis of the convergence and sample complexity of the gradient-based Belief Net is an important open direction. Existing results for stochastic gradient descent on smooth non-convex objectives (Ghadimi and Lan, 2013) guarantee convergence to a stationary point at a rate of $O(1/\sqrt{L})$ after L iterations, but translating these into sample-complexity bounds specific to HMM parameter recovery requires additional structural assumptions (e.g. identifiability and mixing conditions) that are beyond the scope of the present work and are deferred to future research.*

Appendix C. Experiments Results

This section presents extra results that complement the findings reported in Section 4. The results are organized into two main subsections: synthetic HMM data in Section C.1 and real-world text data in Section C.2, corresponding to Section 4.1 and Section 4.2 in the main text, respectively. Each subsection includes additional figures and tables that provide a more comprehensive view of the models' performance across different settings and metrics.

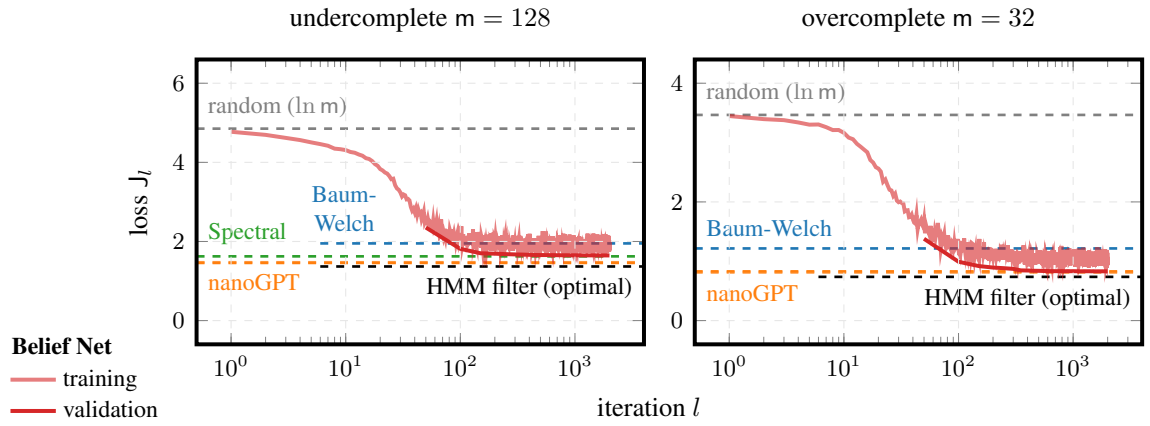

Figure 4: Undercomplete and overcomplete results on synthetic data. The Belief Net's training (light red) and validation (red) loss J_l over iterations l are shown in solid curves. The validation loss is evaluated every 50 iterations. For comparison, horizontal dashed lines show the final validation losses achieved by other methods, including random guess (gray), Baum-Welch (blue), Spectral (green), and nanoGPT (orange).

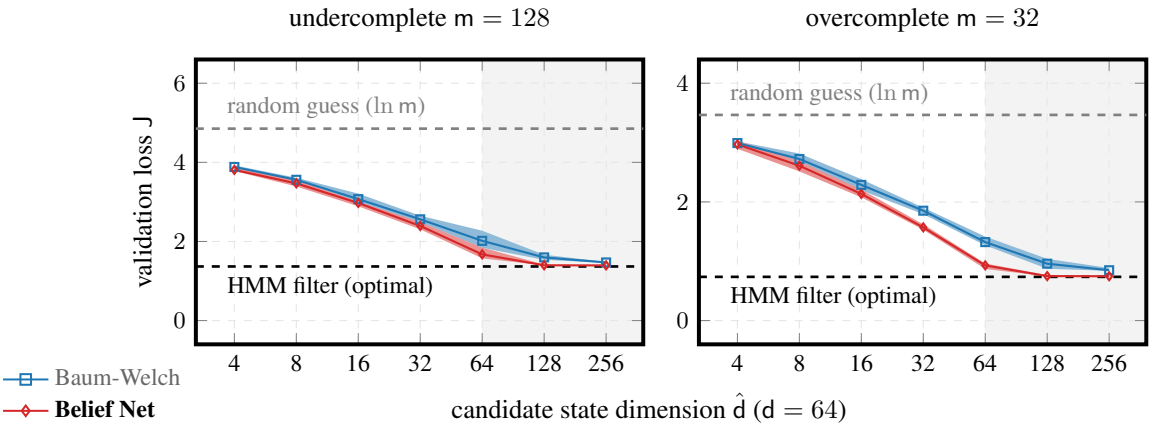

Figure 5: Sensitivity to initialization for the Belief Net (red) and Baum-Welch (blue) methods. Both approaches are evaluated in undercomplete and overcomplete regimes across a range of candidate state dimensions $\hat{d}$. For each setting, 10 independent random initializations are performed, and the resulting validation losses are visualized by their mean (mark), minimum (lower bound), and maximum values (upper bound). The horizontal dashed lines indicate the loss of a random guess (gray) and the optimal HMM filter (black), which represent the worst and best scenarios, respectively.

C.1. Synthetic HMM Data: Prediction Accuracy and Parameter Recovery

C.1.1. Prediction Accuracy

In this setting (see Section 4.1.1), the learner knows both the hidden state dimension d and the observation dimension m. The Belief Net's training and validation loss curves for the undercomplete and overcomplete settings on synthetic HMM data are shown in Figure 4.

To evaluate sensitivity to initialization, we relax the assumption that the hidden state dimension d is known and instead sweep over candidate state dimensions $\hat{d} \in \{4, 8, 16, 32, 64, 128, 256\}$. For each candidate, we assess the variability in validation loss J across multiple random initializations. The results are shown in Figure 5. The Belief Net consistently achieves lower validation loss and exhibits substantially less variability compared to Baum-Welch across all candidate state dimensions. The spectral method is excluded from this analysis, as it does not depend on initialization.

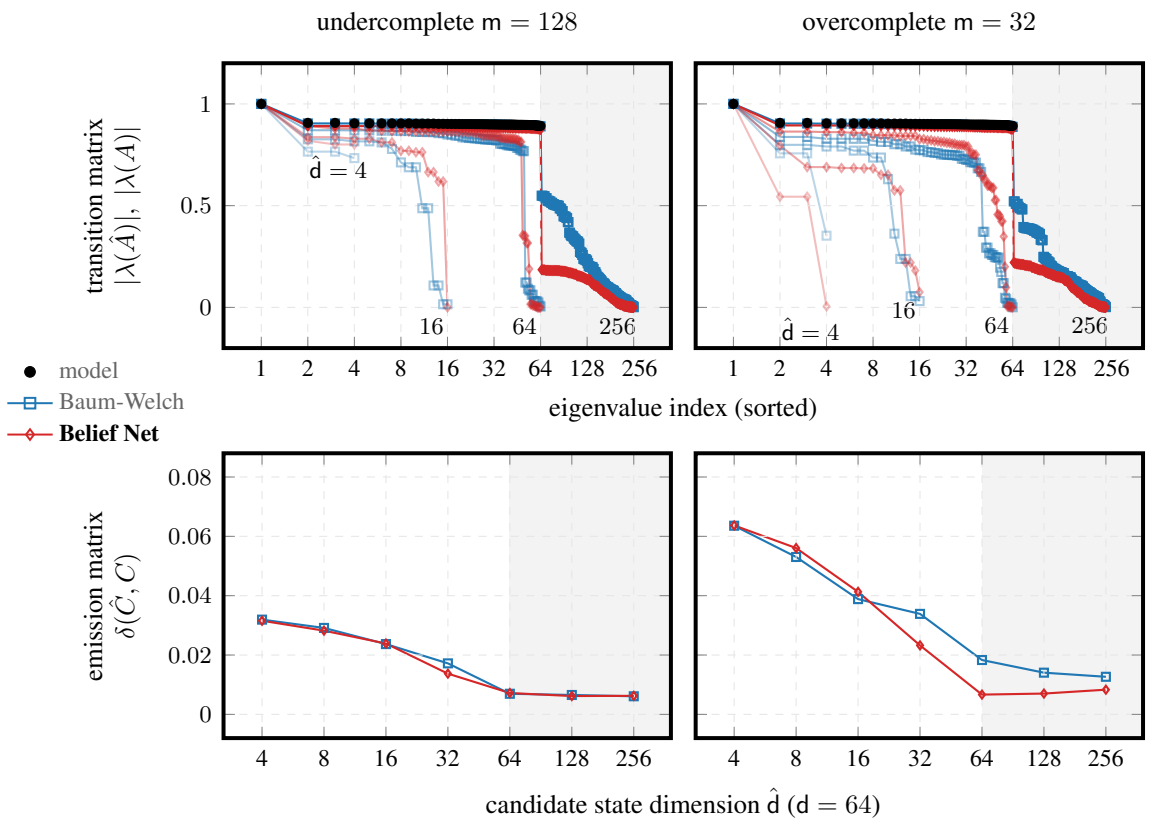

Figure 6: Parameter recovery on synthetic HMM data comparing the Belief Net (red) and Baum-Welch (blue) methods. The top row illustrates the magnitudes of the eigenvalues for the learned transition matrices, arranged in descending order across multiple candidate state dimensions ($\hat{d} \in \{4, 16, 64, 256\}$), where transparency levels differentiate the choices of $\hat{d}$ and black markers denote the true transition matrix eigenvalues for reference. The bottom row evaluates the fidelity of the estimation by reporting the discrepancy between the learned and ground-truth emission matrices as a function of the candidate state dimension $\hat{d}$.

C.1.2. PARAMETER RECOVERY

In this setting (see Section 4.1.2), the learner only knows the observation dimension m. Each model is trained and validated across a range of candidate state dimensions $\hat{d} \in \{4, 8, 16, 32, 64, 128, 256\}$. To evaluate the quality of parameter recovery, we analyze the learned transition and emission matrices from the Belief Net and Baum-Welch methods, where the results are shown in Figure 6. For the transition matrix, we compare the magnitudes of the eigenvalues of the learned transition matrices $|\lambda(\hat{A})|$ to those of the true transition matrix $|\lambda(A)|$, which reflect the system's temporal dynamics and mixing properties. For $\hat{d} > d$, both the Belief Net and Baum-Welch successfully recovers the eigenvalue spectrum of the transition matrix, with the learned d eigenvalues closely matching the true eigenvalues and the remaining $\hat{d} - d$ eigenvalues have a sharp drop in magnitude, indicating that the extra dimensions are effectively ignored. For $\hat{d} \leq d$, the learned eigenvalues deviate from the true eigenvalues, reflecting the model's inability to capture the full dynamics with insufficient state dimensions. For the emission matrix, we evaluate the discrepancy δ between the learned and true emission matrices as a function of the candidate state dimension $\hat{d}$. For a fixed $\hat{d}$, the discrepancy is computed as follows:

$$\delta(\hat{C}, C) = \sum_{k=1}^{m} \bar{p}(z_k) \sum_{i=1}^{\hat{d}} \sum_{j=1}^{d} \gamma_{ij} \left| \hat{C}_{ik} - C_{jk} \right|$$

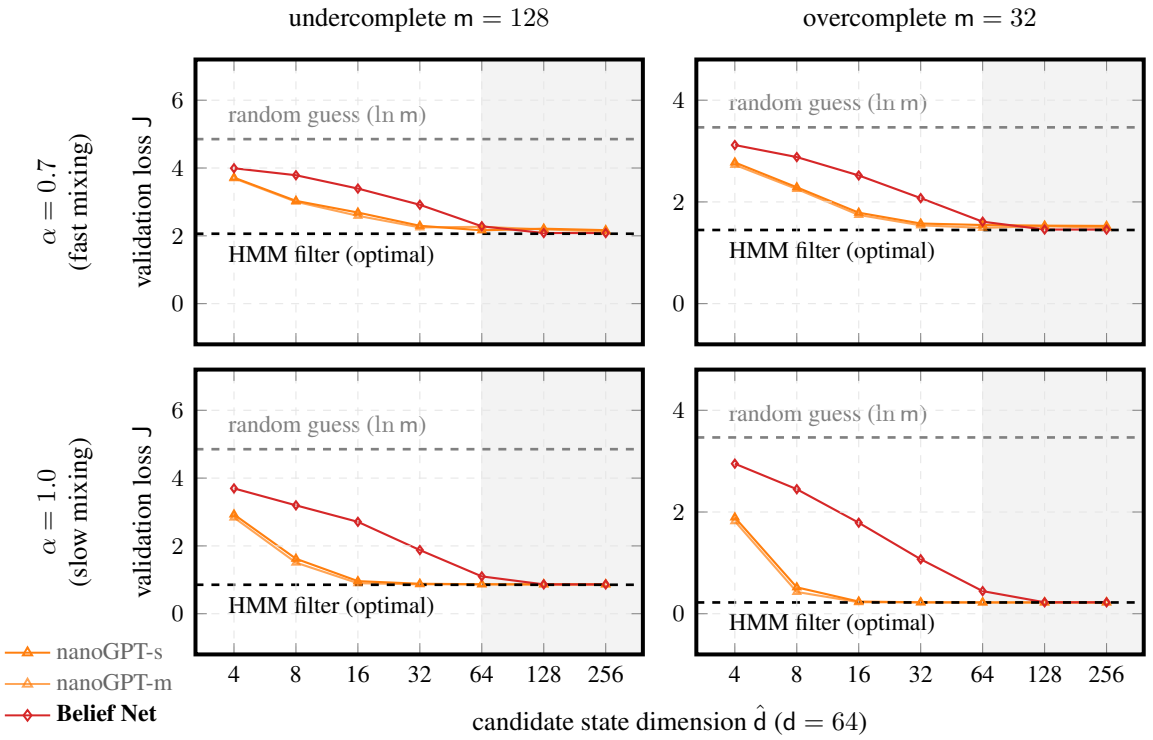

Figure 7: Belief Net (red) v.s. nanoGPTs (orange) on synthetic HMM data with state dimension $\mathsf{d} = 64$. This figure compares the models' validation loss J across candidate state dimensions $\hat{\mathsf{d}}$ for HMM's with both fast and slow mixing, overcomplete and undercomplete settings. Random guess loss (gray) and optimal loss (black) are plotted for worst and best-case comparison.

where $\bar{p} \in \mathcal{P}(\mathbb{O})$ is the stationary distribution of the true HMM's emission process, and $\gamma \in \Gamma := [0, 1/\mathsf{d}]^{\hat{\mathsf{d}}} \times [0, 1/\hat{\mathsf{d}}]^{\mathsf{d}}$ is an optimal coupling between the rows of the learned emission matrix $\hat{C}$ and the true emission matrix C. The coupling γ is computed by solving the following optimal transport problem:

$$\min_{\gamma \in \Gamma} \sum_{i=1}^{\hat{\mathsf{d}}} \sum_{j=1}^{\mathsf{d}} \gamma_{ij} \left\| \hat{C}_{i,:} - C_{j,:} \right\|, \quad \text{s.t.} \sum_{i=1}^{\hat{\mathsf{d}}} \gamma_{ij} = \frac{1}{\mathsf{d}}, \ \forall j = 1, \ldots, \mathsf{d}, \quad \sum_{j=1}^{\mathsf{d}} \gamma_{ij} = \frac{1}{\hat{\mathsf{d}}}, \ \forall i = 1, \ldots, \hat{\mathsf{d}}$$

where the norm $\|\cdot\|$ can be any metric between the rows of the emission matrices. Here, we use the Hellinger distance, which is a common choice for measuring the distance between probability distributions, and the optimal coupling is obtained using the Sinkhorn algorithm (Sinkhorn, 1967) through the Python Optimal Transport (`POT`) library (Flamary et al., 2021). The results show that the Belief Net almost consistently achieves lower emission matrix discrepancy compared to Baum-Welch across all candidate state dimensions $\hat{\mathsf{d}}$, indicating better parameter recovery quality.

In addition to comparing the Belief Net and Baum–Welch methods, we evaluate the Belief Net against nanoGPT-s and nanoGPT-m across varying candidate state dimensions $\hat{\mathsf{d}}$ for both fast- and slow-mixing HMMs. As shown in Figure 7, the nanoGPT models exhibit similar performance across all settings, with closely aligned validation losses, and consistently outperform the Belief Net for all $\hat{\mathsf{d}} \leq \mathsf{d}$. These findings indicate that even single-layer transformers can effectively capture

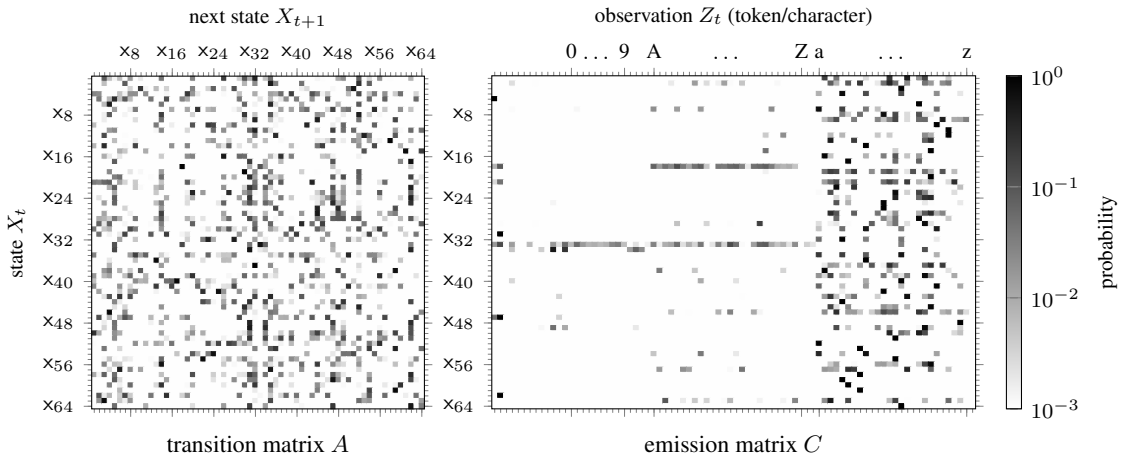

Figure 8: Learned transition and emission matrices of the HMM with Belief Net trained on the real-word text data: the Federalist Papers. The subfigure on the left is the learned transition matrix A. Each entry A_{ij} represents the probability of transitioning from state x_i to state x_j. The color intensity indicates the magnitude of the transition probabilities with darker colors representing higher probabilities. The axes are labeled with the corresponding states. The subfigure on the right is the learned emission matrix C. Each entry C_{ik} represents the probability of emitting token z_k from state x_i. The observations $\mathsf{z}_{14:23}$ correspond to the numeric characters 0-9, $\mathsf{z}_{28:53}$ correspond to the uppercase letters A-Z, and $\mathsf{z}_{56:81}$ correspond to the lowercase letters a-z. The remaining observations correspond to special characters and whitespace. The color intensity indicates the magnitude of the emission probabilities with darker colors representing higher probabilities. The axes are labeled with the corresponding states and tokens. The color bars indicate the log scale of the probabilities with the minimum visualized value set to 10^{-3}.

HMM dynamics, likely due to their ability to model long-range, non-Markovian dependencies, suggesting a potential avenue for improving the Belief Net framework.

C.2. Real-World Text Data: Character-Level Language Modeling

In this setting (see Section 4.2), the Belief Net and all other models are evaluated on a language modeling task for next-token prediction using the Federalist Papers dataset. The trained models include the Belief Net, Baum-Welch, Spectral, nanoGPT-s (single-head single-layer transformer) and nanoGPT-m (multi-head single-layer transformer). Their performance is evaluated in terms of perplexity on the validation set, as shown in Figure 3 in the main text.

The spectral method fails to identify a valid solution for this dataset under the state dimension $\mathsf{d} = 64$, likely due to the non-Markovian nature of the text data. A sweep over the candidate state dimension $\hat{\mathsf{d}} \in [1, \mathsf{d}]$ is performed and the model corresponding to the lowest validation loss is chosen to report the validation perplexity for the spectral method. The final model of the spectral method has $\hat{\mathsf{d}} = 2$ and has the highest perplexity among all methods.

The learned transition and emission matrices of the HMM with Belief Net are visualized in Figure 8. The transition matrix A shows that each state only transitions to a few other states, indicating that the model is learning a sparse transition structure. The emission matrix C shows more interpretable patterns: certain states emits specific characters with high probability. For example, state x_{33} emits the digits and uppercase letters with high probability, while state x_{18} emits uppercase letters only. The likelihood of emitting lowercase letters is generally high across all states, which may be due to the fact that the dataset contains more lowercase letters than all other characters.