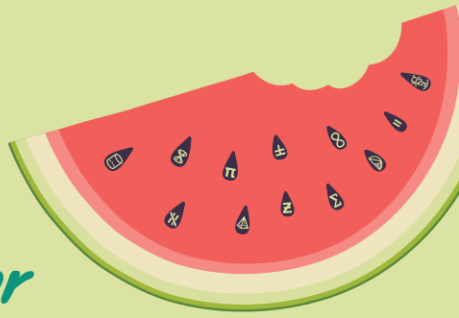
AMSI SUMMER RESEARCH SCHOLARSHIPS 2024–25

Get a taste for Research this Summer



Deep Learning in Hidden Markov Models

Kyan Percevault

Supervised by Lewis Mitchell, Matthew Roughan and Angus Lewis

University of Adelaide

28/02/2025

Jane Street®

AMSI AUSTRALIAN MATHEMATICAL SCIENCES INSTITUTE

Abstract

Hidden Markov Models (HMMs) are widely used to model discrete-time processes. However, given a sequence of observations, inference of suitable parameters for an HMM is difficult. We investigate the potential for Recurrent Neural Networks (RNNs) in parameter inference, and compare the results to the Baum-Welch algorithm. Although all loss functions that we derive are limited by computational costs or state permutations, experimental results indicate that the performance of RNNs is comparable to the Baum-Welch algorithm, at least in the case of 2 hidden states. We hope that these promising results will motivate further research into permutation-invariant loss functions and more complicated deep-learning applications in HMM parameter inference. We also review existing approaches when the number of hidden states is unknown, and we highlight the difficulties of deep-learning approaches in this case.

1 Introduction

Hidden Markov Models (HMMs) have been applied in fields such as speech recognition, network analysis and bioinformatics [8]. The typical approach to HMM parameter inference is the Baum-Welch algorithm, although this algorithm is susceptible to getting stuck in local minima [12]. As an alternative, in this project we investigate the potential of deep-learning models for parameter inference, namely Recurrent Neural Networks (RNNs) and hybrid RNN/Baum-Welch architectures.

We first derive loss functions for model training based on Dirichlet distributions (parametrised by model outputs), Jensen-Shannon Divergence between corresponding distributions and sequence likelihoods under the inferred parameters. We identify that the number of operations and permutations of state labels are significant limitations for these loss functions. Nonetheless, we implement an RNN architecture which can handle an unknown number of sequences of varying lengths, which can be used for parameter inference directly or as initialisation for the Baum-Welch algorithm. Empirical comparison shows that these deep-learning models produce results which are comparable to the Baum-Welch algorithm.

Finally, we reviewed some existing approaches when the number of hidden states is unknown, namely the use of information criteria alongside Baum-Welch and the Hierarchical Dirichlet Process HMM. We also provide a high-level description of potential deep-learning techniques when the number of states is unknown, although we highlight the high computational costs of these approaches.

Statement of Authorship

Kyan Percevault produced all results and interpretations used in this report, as well as all associated code. Professor Lewis Mitchell, Professor Matthew Roughan and Angus Lewis provided general direction, motivation and some of the references used throughout this report. Professor Lewis Mitchell and Angus Lewis also proofread this report.

2 Background

A Hidden Markov Model (HMM) is a generative model where a latent state is governed by a Markov chain, and the distribution of observations is determined by the latent state at that time step. For a single set of HMM parameters, We will denote the set of associated observation sequences with $\boldsymbol{o}$ and the sequences of corresponding hidden states with $\boldsymbol{h}$; then, the hidden state and observation at time t can be denoted h_t and o_t, respectively. As described in [12], an HMM is characterised by the quintuplet (N, M, A, B, π), where:

- N is an integer for the number of hidden states, with $S = \{s_1, s_2, ..., s_N\}$ the set of distinct state labels.
- M is the number of distinct observations, with $V = \{v_1, v_2, ..., v_M\}$ the set of distinct observations.
- $A = \{a_{ij}\}$ is the transition probabilities of the hidden Markov chain, $a_{ij} = \Pr(h_{t+1} = s_j | h_t = s_i) \quad \forall 1 \leq i, j \leq N$.
- $B = \{b_j(k)\}$ is the emission distributions, $b_j(k) = \Pr(o_t = v_k | h_t = s_j) \quad \forall 1 \leq j \leq N,\ 1 \leq k \leq M$.
- $\pi = \{\pi_i\}$ is the initial distribution of states, $\pi_i = \Pr(q_1 = v_i) \quad \forall 1 \leq i \leq N$.

Once the parameters are known, HMMs can be used to calculate the most likely sequence of latent states from an observation sequence (via the Viterbi algorithm), evaluate the likelihood of a given observation sequence (via the Forward Algorithm), or generate sequences of observations and corresponding latent states [12]. However, parameter inference (that is, inferring the most likely parameters of an HMM given a set of observation sequences) remains a significant challenge. Typical approaches such as Baum-Welch can converge to incorrect local minima for log-likelihood, and hence an accurate initialisation of parameters is necessary [5].

2.1 Baum-Welch Algorithm

The Baum-Welch algorithm is a special case of the Expectation-Maximisation (E-M) algorithm. This is a useful approach when part of the data is hidden, but we wish to maximise the log-likelihood of both the hidden data, $\boldsymbol{h}$, and the observed data, $\boldsymbol{o}$ [15]. The algorithm is a 2-step process, where we first create an auxiliary function of the model parameters λ by taking the expectation over the hidden data, conditioned on the observed data and the previous model parameters, λ_0:

$$Q(\lambda|\lambda_0) = \int \Pr(\boldsymbol{h}|\boldsymbol{o}; \lambda_0) \log \Pr(\boldsymbol{o}, \boldsymbol{h}; \lambda) d\boldsymbol{h}$$

Then, in the second step, we maximise the auxiliary function, thereby obtaining the best new value of the parameters from their previous values [15]. Log-likelihood of data (hidden and observed) is guaranteed to be non-decreasing in the Expectation-Maximisation algorithm: $\Pr(\boldsymbol{o}|\lambda_{k+1}) \geq \Pr(\boldsymbol{o}|\lambda_k)$ In fact, we are guaranteed to converge to a critical point for the sequence likelihood [12], although this point may not be the global optima. A common approach to increase the chance of convergence to the globally-optimal parameters is to apply the Baum-Welch algorithm several times with different initial parameters, although this is computationally inefficient.

3 Deep Learning

In this section, we provide an overview of deep learning and describe the main deep-learning technique used in this report: RNNs.

3.1 Supervised Learning Overview

In supervised learning, our goal is to learn some mapping between inputs and target features. This involves a training dataset, containing inputs and their corresponding labels for the target features, and a loss function (for example, Mean Square Error) which quantifies the difference between the output of our machine-learning model and the true label. Then, we "fit" our model by finding the model parameters which minimise the loss over our training set. We typically do this by iteratively updating parameters using gradient-based optimisers such as Stochastic Gradient Descent (SGD) or Adam [11].

3.2 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a deep-learning model which account for order in a set of inputs, thereby making it an appropriate choice to fit HMM parameters from observation sequences. Further, RNNs allow inputs of varying length, which is necessary when the length of the observation sequences are unknown. In RNNs, the hidden state is a vector of real numbers which is transferred between time steps. At each time step, *neurons* (also called perceptrons, nodes or activation units) are used to update the hidden state based on the current observations and the previous hidden state. Inside each neuron, a linear combination is taken of all inputs, a constant bias is added, then a non-linear function is applied (typically ReLU, hyperbolic tan, or the sigmoid function [11]). Since each neuron produces a scalar, we can stack multiple neurons (each with independent weights and biases) to produce a vector.[1] In the case of RNNs, the neurons at each time step produce a non-linear map to update the hidden state with information from the observation at that time step [11].

We utilise a many-to-one RNN, as observation sequences span multiple time steps, but we require only a single vector of parameters as output; therefore, only the hidden state at the final time step is necessary to compute outputs. To ensure that the output has the correct length, we apply a final fully-connected layer. Once again, this involves taking a linear combination of the hidden state and adding a bias, and we repeat this for the required number of HMM parameters. Finally, we apply the softmax function,

$$\sigma(\boldsymbol{z})_i = \frac{e^{z_i}}{\sum_j e^{z_j}},$$

to different sections of the output vector. This ensures that each distribution in the output satisfies the probability constraints: elements must be non-negative and sum to 1. Specifically, we apply softmax to the first

[1] It is convention to stack the weights corresponding to each input type into matrices. Specifically, W_{hh} and W_{xh} represent the weights across all neurons which are assigned to the previous hidden state and the current observation (respectively) to update the hidden state.

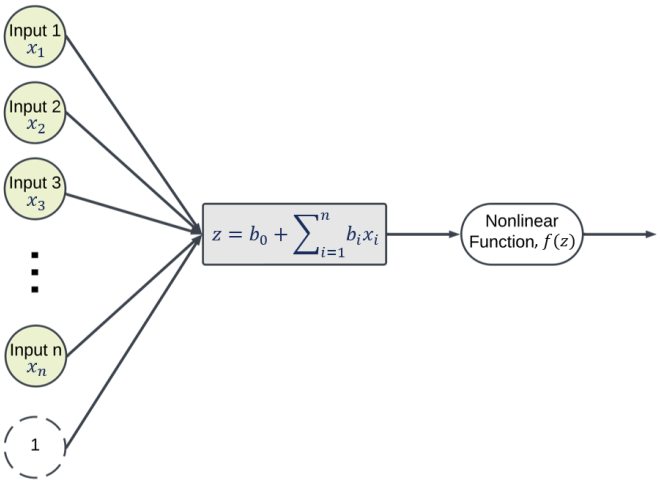


Figure 1: A diagram of a deep-learning neuron. In RNNs, some of the inputs will be from the previous hidden state, and some will be from the observations at that time step.

N components (initial probabilities), then the next N groups of N components (transition probabilities), and finally to the next N groups of M components (emission probabilities).

Vanishing and exploding gradients are common issues in RNN implementation. Vanishing gradients prevent the RNN from retaining information over a large number of time steps [10], while exploding gradients cause difficulty training since gradient descent updates may produce worse parameters [1]. Consider the case when the derivative of the activation function, f', is bounded such that $||\text{diag}(f'(z_k))|| \leq \gamma$ for every element of the hidden state, z_k. Then, $\lambda^* < \frac{1}{\gamma}$ is a sufficient condition for vanishing gradients (and therefore a necessary condition for exploding gradients is $\lambda^* > \frac{1}{\gamma}$), where λ^* is the absolute value of the largest eigenvalue in the hidden-to-hidden weight matrix W_{hh} [10]. However, it is also necessary to avoid complicated loss functions which apply many operations to the inputs or outputs, as this will also increase susceptibility to numerical instability of gradients

Additionally, how do we determine suitable values for hyperparameters such as batch size, learning rate, optimiser, and hidden size? And in the case of training RNNs for HMM parameter inference, the datasets are completely synthetic, so we have configurative parameters in the sizes of the training, validation and test datasets. In this report, we will keep the batch size at 16, while training/validation/testing dataset sizes were kept at 300/50/100 (as these kept runtime below 12 hours). However, the learning rate, optimiser and hidden size were optimised with Optuna, a Python package which utilises search algorithms and pruning to efficiently trial different hyperparameter values.[2]

[2]https://optuna.org/

4 Parameter Inference with a Known Number of States

We mainly focus on the scenario where the number of states in the HMM is known. Derivation of loss functions, development of RNN architectures and empirical comparison to Baum-Welch will occur in the following sections.

4.1 Loss Functions with a Known Number of States

In this section, 3 loss functions specific to parameter inference for HMMs are derived, and the practical issues associated with each are discussed.

4.1.1 Dirichlet Distribution Parametrisation

Let an arbitrary set of inputs and targets be denoted $\boldsymbol{o}$ and $\boldsymbol{y}$, respectively. In our case, $\boldsymbol{o}$ is a set of observation sequences and $\boldsymbol{y}$ is the concatenation of the flattened initial probabilities $\boldsymbol{\pi}$, transition probabilities A and emission probabilities B. According to [11], a loss function is typically derived by first selecting a parametric distribution for the outputs, $\Pr(\boldsymbol{y}|\boldsymbol{\theta})$. Then, instead of producing the "best guess" for the outputs directly, deep-learning models should be configured to produce a set of output-parameters $\boldsymbol{\theta}^*$, and we select the model-parameters which maximise the probability of the true labels under the distributions produced, $\Pr(\boldsymbol{y}|\boldsymbol{\theta}^*)$. Letting our model be denoted $\boldsymbol{f}(\boldsymbol{x}|\boldsymbol{\phi})$, the optimal model-parameters $\hat{\boldsymbol{\phi}}$ will therefore maximise the likelihood $\Pr(\boldsymbol{y}|\boldsymbol{f}(\boldsymbol{x}|\boldsymbol{\phi}))$. Assuming the sets of parameters $\boldsymbol{y}_1, \boldsymbol{y}_2, ..., \boldsymbol{y}_D$ are independent and identically distributed, we obtain the maximum likelihood criterion

$$\hat{\boldsymbol{\phi}} = \arg\max_{\boldsymbol{\phi}} \left[\prod_{i=1}^{D} \Pr(\boldsymbol{y}_i \mid \boldsymbol{f}(\boldsymbol{x}_i \mid \boldsymbol{\phi})) \right].$$

We can equivalently minimise the negative log of this criterion:

$$\hat{\boldsymbol{\phi}} = \arg\min_{\boldsymbol{\phi}} \left[-\sum_{i=1}^{D} \log \Pr(\boldsymbol{y}_i \mid \boldsymbol{f}(\boldsymbol{x}_i \mid \boldsymbol{\phi})) \right]. \tag{1}$$

Therefore, as explained by [11], we can construct a loss function $\mathcal{L}(\boldsymbol{\phi})$ from Equation 1:

$$\mathcal{L}(\boldsymbol{\phi}) = -\sum_{i=1}^{D} \log \Pr(\boldsymbol{y}_i \mid \boldsymbol{f}(\boldsymbol{x}_i \mid \boldsymbol{\phi})), \tag{2}$$

$$\text{where} \quad \hat{\boldsymbol{\phi}} = \arg\min \mathcal{L}(\boldsymbol{\phi}) \tag{3}$$

Since the parameters of an HMM consist of multiple distributions, Dirichlet distributions may be appropriate for $\Pr(\boldsymbol{y}|\boldsymbol{\theta})$. A Dirichlet distribution with parameters $\boldsymbol{\beta} = (\beta_1, \beta_2, ..., \beta_k)$ has support on all categorical distributions of length k [4], making it a "distribution over distributions". In our case, we can split the ith set of true parameters into a vector of distributions, $\boldsymbol{y}_i = (\boldsymbol{y}_{i,1}, \boldsymbol{y}_{i,2}, ..., \boldsymbol{y}_{i,2N+1})$, where $\boldsymbol{y}_{i,j}$ is the jth distribution within the ith set of parameters (the initial, transition, and emission probabilities of the HMM contribute 1, N, and N distributions, respectively). Similarly, the parameters of the jth Dirichlet distribution within the ith set of parameters can be denoted $\boldsymbol{\beta}_{i,j}$. Then, we can divide the outputs among distributions, which we will represent with $\boldsymbol{f}(\boldsymbol{o_i}|\boldsymbol{\phi}) = (\boldsymbol{f}_{i,1}, ..., \boldsymbol{f}_{i,2N+1}) = (\boldsymbol{\beta}_{i,1}, ..., \boldsymbol{\beta}_{i,2N+1})$, where $\boldsymbol{f}_{i,j} = \boldsymbol{f}_{i,j}(\boldsymbol{o}_i|\boldsymbol{\phi})$ is the output corresponding

to the jth distribution of the ith set of sequences.[3] [4] Assuming that each output distribution is independent, Equation 2 becomes

$$\mathcal{L}(\boldsymbol{\phi}) = -\sum_{i=1}^{D} \log \Pr\left((\boldsymbol{y}_{i,1}, ..., \boldsymbol{y}_{i,2N+1}) \mid (\boldsymbol{f}_{i,1}, ..., \boldsymbol{f}_{i,2N+1})\right) \tag{4}$$

$$= -\sum_{i=1}^{D} \sum_{j=1}^{2N+1} \log \Pr\left(\boldsymbol{y}_{i,j} | \boldsymbol{f}_{i,j}\right). \tag{5}$$

We can apply the probability density function of a Dirichlet distribution to Equation 5. Let

$$\ell_j = \begin{cases} N, & \text{if } j \in \{1, ..., N+1\} \\ M, & \text{if } j \in \{N+2, ..., 2N+1\} \end{cases}$$

be the length of the jth distribution, and let $f_{i,j,k}$ and $y_{i,j,k}$ be the kth element in the jth distribution of the ith data point of model outputs and true parameters, respectively. Then,

$$\Pr(\boldsymbol{y}_{i,j} | \boldsymbol{\beta}_{i,j}) = \frac{\Gamma(\sum_{k=1}^{\ell_j} \beta_{i,j,k})}{\prod_{k=1}^{\ell_j} \Gamma(\beta_{i,j,k})} \prod_{j=1}^{k} y_{i,j,k}^{\beta_{i,j,k}-1}, \tag{6}$$

hence our final loss function is

$$\mathcal{L}(\boldsymbol{\phi}) = -\sum_{i=1}^{D} \sum_{j=1}^{2N+1} \left[\log \Gamma\left(\sum_{k=1}^{\ell_j} f_{i,j,k} \right) - \sum_{k=1}^{\ell_j} \log \Gamma(f_{i,j,k}) + \sum_{k=1}^{\ell_j} (f_{i,j,k} - 1) \log y_{i,j,k} \right].$$

4.1.2 Log-Likelihood

Another justifiable loss function is log-likelihood, as it is natural to attempt to find the HMM parameters which maximise the likelihood of the given observation sequences. Further, this loss function allows easy comparison with the Baum-Welch algorithm. The log-likelihood can be easily calculated from HMM parameters using the forward algorithm (see [12]).

4.1.3 Jensen-Shannon Divergence (JSD)

Finally, loss could be computed by measuring the element-wise or distribution-wise divergence between the fitted and true HMM parameters. One option is KL-divergence, which is defined by $D_{KL}(P||Q) = \sum_x P(x) \log\left(\frac{P(x)}{Q(x)}\right)$. If P is the true distribution, then KL-divergence is a measure of the inefficiency of approximating P with the distribution Q [3]. In theory, this makes $D_{KL}(\boldsymbol{y}_{i,j}||\boldsymbol{f}_{i,j})$ a logical choice to compare the true and fitted distributions, although an issue arises when $f_{i,j,k} = 0$ and $y_{i,j,k} \neq 0$ anywhere, as this makes KL-divergence undefined.[5] We should allow for the inferred probability to be 0 but not the true probability, or vice versa. This motivates

[3] To satisfy the positivity requirement for the parameters of a Dirichlet distribution, we apply the function $\text{softplus}(x) = \log(1+e^x)$ to outputs of the model before computing loss.

[4] When we use our deep-learning model for inference, we must convert our distribution $\Pr(\boldsymbol{y}|\boldsymbol{\theta})$ to a point estimate for $\boldsymbol{y}$, typically by selecting the distribution mode [11]. If $\boldsymbol{y}_{i,j} = (y_{i,j,1}, y_{i,j,2}, ..., y_{i,j,\ell_j}) \sim \text{Dir}(\boldsymbol{\beta}_{i,j})$, the mode of $\Pr(\boldsymbol{y}_{i,j}|\boldsymbol{\beta}_{i,j})$ is at $y_{i,j,k} = \frac{\beta_{i,j,k}-1}{\sum_{\kappa=1}^{\ell_j} \beta_{i,j,\kappa} - \ell_j} \quad \forall k = 1, 2, ..., \ell_j$.

[5] There is no issue when $P(x) = 0 = Q(x)$, as the contribution of this term is set to 0 since $\lim_{x \to 0^+} x \log(x) = 0$

the use of Jensen-Shannon Divergence (JSD), which is a version of KL-Divergence which is symmetric, bounded and defined everywhere [2]. JSD is given by

$$D_{JS}(\boldsymbol{f}_{i,j}||\boldsymbol{y}_{i,j}) = \frac{1}{2}D_{KL}\left(\boldsymbol{f}_{i,j} \,||\, \frac{1}{2}\left(\boldsymbol{f}_{i,j}+\boldsymbol{y}_{i,j}\right)\right) + \frac{1}{2}D_{KL}\left(\boldsymbol{y}_{i,j} \,||\, \frac{1}{2}\left(\boldsymbol{f}_{i,j}+\boldsymbol{y}_{i,j}\right)\right).$$

Therefore, JSD is the sum of KL-divergences to the average of $\boldsymbol{f}_{i,j}$ and $\boldsymbol{y}_{i,j}$. JSD is the square of a metric [9], meaning $\sqrt{D_{JS}}$ is symmetric, positive, 0 only if two inputs are equal, and obeys the triangle inequality.

4.1.4 Discussion of Loss Functions

For any loss function which involves comparison between corresponding elements or distributions, the ordering of states must be considered. Each state is arbitrarily assigned an integer label, but the order of these labels has no bearing on the functionality of the HMM. For example, the two following sets of parameters are equivalent (from swapping the order of the first and second states).

$$\begin{aligned} \boldsymbol{\pi} &= \begin{bmatrix} 0.1 & 0.9 \end{bmatrix}, \\ A &= \begin{bmatrix} 0.25 & 0.75 \\ 0.4 & 0.6 \end{bmatrix}, \\ B &= \begin{bmatrix} 0.2 & 0.3 & 0.5 \\ 0.1 & 0.3 & 0.6 \end{bmatrix}, \end{aligned} \qquad \begin{aligned} \boldsymbol{\pi} &= \begin{bmatrix} 0.9 & 0.1 \end{bmatrix}, \\ A &= \begin{bmatrix} 0.6 & 0.4 \\ 0.75 & 0.25 \end{bmatrix}, \\ B &= \begin{bmatrix} 0.1 & 0.3 & 0.6 \\ 0.2 & 0.3 & 0.5 \end{bmatrix}. \end{aligned}$$

The approach suggested by [7] is to normalise an HMM by assigning State i the score $\sum_{k=1}^{M} b_i(k) \cdot k$, then labelling states according to the ascending order of these scores. However, this is not a perfect solution. Consider the following example, where A^*, B^* are the true transition and emission probabilities of an HMM and $A_1, B_1,\ A_2, B_2$ are candidates to approximate these parameters:

$$\begin{aligned} A_1 &= \begin{bmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{bmatrix}, \\ B_1 &= \begin{bmatrix} 0.51 & 0.49 \\ 0.48 & 0.52 \end{bmatrix}, \end{aligned} \qquad \begin{aligned} A^* &= \begin{bmatrix} 0.99 & 0.01 \\ 0.01 & 0.99 \end{bmatrix}, \\ B^* &= \begin{bmatrix} 0.49 & 0.51 \\ 0.48 & 0.52 \end{bmatrix}, \end{aligned} \qquad \begin{aligned} A_2 &= \begin{bmatrix} 0.99 & 0.01 \\ 0.01 & 0.99 \end{bmatrix}, \\ B_2 &= \begin{bmatrix} 0.47 & 0.53 \\ 0.48 & 0.52 \end{bmatrix}. \end{aligned}$$

The candidates A_2, B_2 are clearly better than A_1, B_1, as $A_2 = A^*$ exactly and $B_2 \approx B^* \approx B_1$, although A_1 and A^* are significantly different. However, after permuting the states based on the rule described above, A_2 and B_2 must have the state order reversed, resulting in the permuted matrices

$$\tilde{A_2} = \begin{bmatrix} 0.01 & 0.99 \\ 0.99 & 0.01 \end{bmatrix}, \quad \tilde{B_2} = \begin{bmatrix} 0.52 & 0.48 \\ 0.53 & 0.47 \end{bmatrix}.$$

The other parameters will not be permuted, so $\tilde{A^*} = A^*$, $\tilde{B^*} = B^*$, $\tilde{A_1} = A_1$, and $\tilde{B_1} = B_1$. Clearly, the second set of parameters is now much closer to the true parameters, despite being significantly worse before applying the permutation.

The above example illustrates how the scaling in [7] can sometimes place too much emphasis on emission probabilities. A more thorough alternative would be to compute the loss for every single state permutation

and choose the permutation with the lowest loss. However, this is computationally prohibitive since it requires loss to be computed $N!$ times per data point. The example above also shows that small changes to the emission probabilities can produce significant changes in the parameters produced, hence permutations will produce discontinuities in the loss landscape. For these reasons, any loss function which compares corresponding parameters or distributions (Dirichlet likelihood, JSD, MSE, etc.), will not be suitable. Although, some empirical evidence of the impact of this permutation issue in-practice can be seen in Figure 2. Where the number of states is $N = 2$, the HMM normalisation does not perform much worse than trialling all state permutations, suggesting element-wise and distribution-wise losses may still be viable in this case.

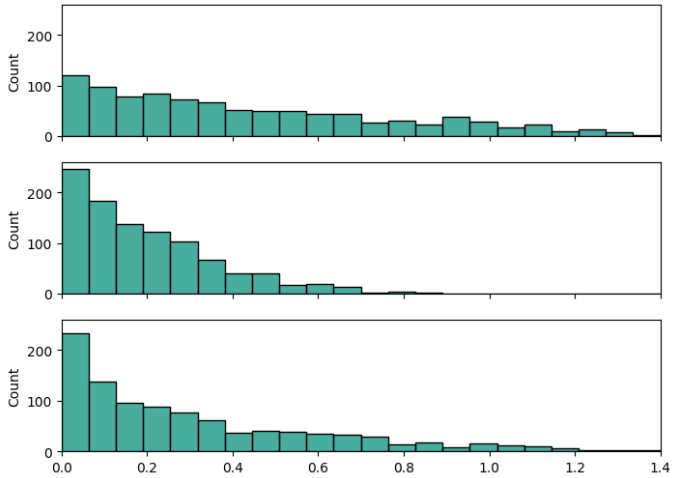


Figure 2: MSE between the true parameters and those fitted with Baum-Welch, $N = 2, M = 3$. TOP: no permutation applied. MIDDLE: all permutations applied, and lowest MSE taken. BOTTOM: HMM Normalised as per [7]. Normalisation performs well compared to all permutations.

Another consideration is the number of computations required to evaluate the loss. As explained in Section 3.2, we wish to use a computationally simple loss function to minimise susceptibility to exploding and vanishing gradients. Therefore, we consider how the number of operations in our loss function scales with the number of states (N) and the total number of observations ($\sum_{j=1}^{n_o} T_j$, where T_j is the length of the jth observation sequence for a particular set of HMM parameters and n_o is the number of sequences). It can be shown that the number of operations to compute JSD and Dirichlet loss is $O(N^2)$. Contrastingly, for negative log-likelihood, the forward algorithm necessitates $O(N^2)$ operations at each time step in each sequence, hence the number of operations is $O(N^2 \sum_{j=1}^{n_o} T_j)$. This is a significant limitation of negative log-likelihood; when the computations scale (even linearly) with the total number of observations, loss calculations quickly become infeasible for RNNs. When this was implemented in PyTorch, training failed due to vanishing or exploding gradients in every trial, even when a single sequence of 10-20 observations was used.

The loss functions derived in this section clearly have their shortfalls. Nonetheless, we implement deep-learning models in the following sections, using JSD as the loss function for training. This will provide a rough

indication of the suitability of deep learning in HMM parameter inference, and whether it is worthwhile to pursue another loss function which is computationally viable and invariant under state permutations.

4.2 RNN and Hybrid Architectures

As mentioned in Section 3.2, RNNs allow inputs of variable lengths. Therefore, an RNN can be applied directly to learn the parameters of an HMM when there is a single observation sequence. It is not as straightforward when there is an unknown number of sequences, as the number of inputs at each time step determines the shape of one of the weight matrices in the RNN. While it would be possible to train a single RNN which allows (up to) an extremely large number of sequences, this would be highly computationally inefficient and would place an upper limit on the number of permitted sequences.

A more elegant solution is to pass each sequence through the same RNN separately, then aggregate the hidden states from each sequence. A single neural network and piecewise-softmax are then applied to the hidden state in the same way as a "regular" RNN. Ideally, the learned weights cause the hidden state to generally increase in magnitude over time. Therefore, longer sequences, which will generally contain more information about the underlying HMM parameters, can contribute more greatly to the post-softmax output of the RNN. A diagram of this architecture can be seen in Figure 3 (b).

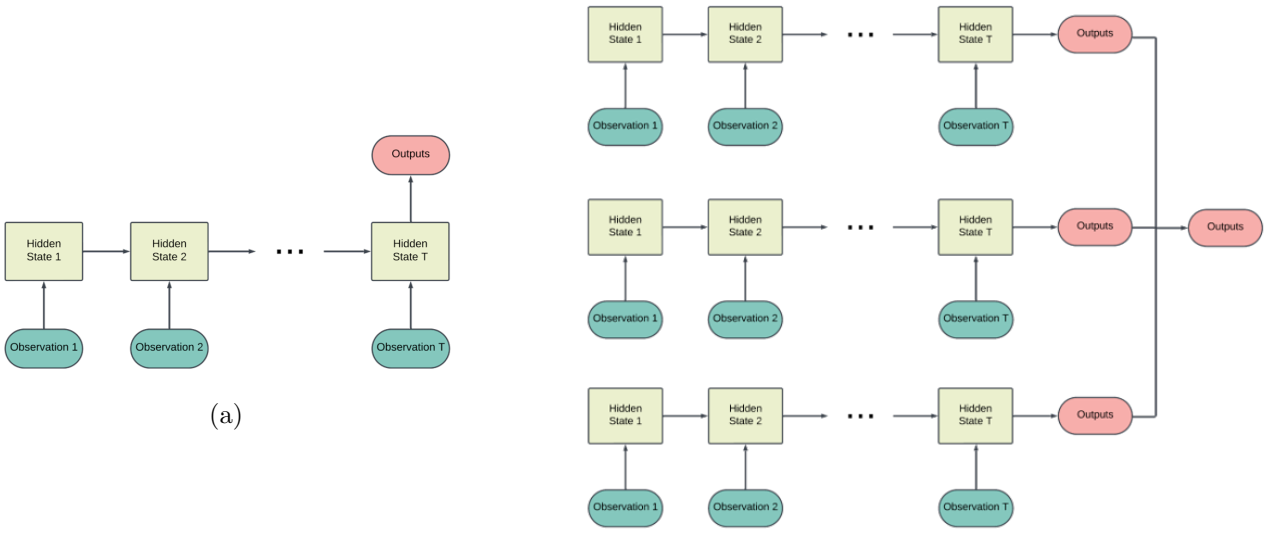


Figure 3: (a) A "regular" RNN. (b) To allow for an unknown number of sequences, we pass each observation sequence through the same "regular" RNN, separately.

An alternative hybrid architecture is to use the result of the RNN above as initialisation for the Baum-Welch algorithm. We will call this the Baum-Welch initialisation model. Conceptually, the RNN should produce a set of parameters which is close enough to the true values so that the Baum-Welch algorithm converges to the global maximum likelihood estimator, rather than a local equivalent. A potential benefit of this model is that

we do not require as many iterations in the Baum-Welch algorithm, as the parameters should begin close to the true values. Having fewer Baum-Welch iterations will minimise the additional computational costs relative to the standalone RNN model.

Originally, loss from the RNN model was calculated after applying the Baum-Welch algorithm, as this allowed the behaviour of Baum-Welch to be "learned" by the model. For example, the RNN may learn to produce parameters in a particular region of the loss landscape in which the Baum-Welch algorithm converges to the global optimum, rather than simply the region with the smallest loss before applying the Baum-Welch algorithm. However, just as with sequence likelihoods (Section 4.1.2), the linear scaling with total observation lengths caused the model to consistently fail due to vanishing and exploding gradients. Instead, the loss of the RNN was kept separate from the Baum-Welch algorithm, meaning that the pre-trained RNN model could be used directly.

4.3 Discussion of Experimental Results

To allow comparison between the Baum-Welch algorithm, RNN and hybrid models, we use ensemble learning as a deep-learning equivalent to multiple parameter initialisations in the Baum-Welch algorithm. To compare to n initialisations, we generated outputs from the top-n performing models from the Optuna hyperparameter search (60 trials), then averaged the results across all models. This is motivated by the rough computational equivalence between the evaluation of multiple trained models and initialisations in the Baum-Welch algorithm, as both lead to an n-times increase in computational requirements. By using ensemble learning with different hyperparameters, we hope to increase generalisability of the RNN; this technique is commonly applied in the subfield of automated-Machine learning [14].

We use JSD as a loss function for training, and both JSD and negative log-likelihood for evaluation (all plots are shown in Appendix A). As evident in Figures 9 and 10, the Baum-Welch initialisation model performs the best from the perspective of negative log-likelihood when there are multiple observation sequences, while still having comparable JSD losses in this case. However, the initialisation model sometimes produces extremely large JSD loss when a single observation sequence is used, which can be seen in Figure 4. This may be caused by large change in parameters during the first few iterations of the Baum-Welch algorithm, which improve log-likelihood but significantly increase JSD. Since the Baum-Welch algorithm does not have any outliers, these large changes seem to be corrected in later iterations when the algorithm is permitted to continue. We will leave further investigation of this behaviour as an avenue for future research, so we will instead compare the RNN and Baum-Welch models.

From Figures 11, 12, 13 and 14, the RNN generally performed slightly better for JSD loss, while the Baum-Welch algorithm resulted in higher log-likelihood (which is to be expected, since Baum-Welch converges to a local minima for sequence likelihood [12]). However, both approaches had highly similar results. Interestingly, the Figures in Appendix A show that changes to the number of initialisations and lengths of observation sequences has virtually no impact on the resultant losses.

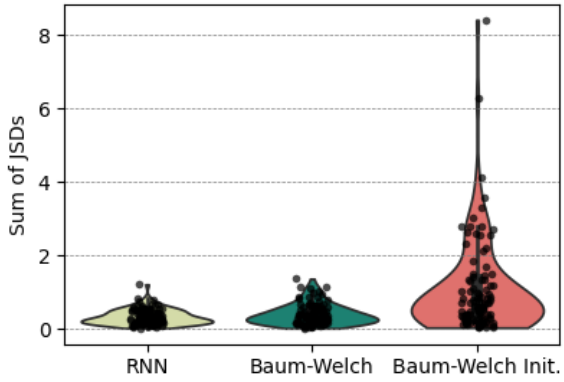
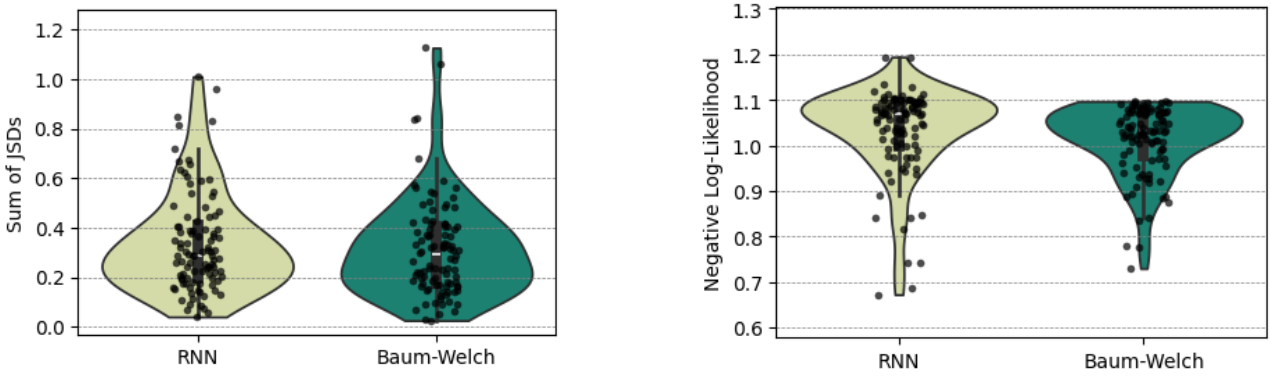


Figure 4: The performance of RNNs trained with JSD (light green), RNN Baum-Welch Initialisation (red) and Baum-Welch (dark green). 5-10 sequences of length 10-20, and 10 initialisations. Evaluated with JSD.



(a) (b)

Figure 5: The performance of RNNs trained with JSD (light green), and Baum-Welch (dark green). 5-10 sequences of length 10-20 and 10 initialisations. Evaluated with (a): JSD and (b): negative log-likelihood.

Considering the issues with JSD as a loss function (see Section 4.1.3) and that a simple RNN is being used, these results are promising. Perhaps this provides motivation for further research to develop a permutation-invariant and computationally viable loss function, and to apply more advanced deep-learning techniques such as Long Short Term Memory (LSTM).

5 Parameter Inference with an Unknown Number of States

The scenario where the number of states is unknown beforehand is significantly more difficult. Here, we introduce existing approaches, Baum-Welch with information criteria and the Hierarchical Dirichlet Process Hidden Markov Model (HDP-HMM), and provide an outline of potential loss functions and deep-learning models.

5.1 Loss Functions with an Unknown Number of States

All loss functions in Section 4.1 are invalid if the number of states is unknown, as the size of the HMM parameters is variable and the likelihood of the given observation sequences will be heavily influenced by the number of states.[6]

Rather than defining an HMM by the parameters, we can define an HMM by the probability distribution of all possible sequences generated. To evaluate similarity we can generate a large number of sequences $\boldsymbol{o}^{(1)}$ from model $\lambda^{(1)}$, calculate the log-likelihood of all of these, then compare these to the log-likelihood under another model $\lambda^{(2)}$ [12].

$$D(\lambda^{(1)}, \lambda^{(2)}) = \frac{1}{T}(\log \Pr(\boldsymbol{o}^{(1)}|\lambda^{(1)}) - \log \Pr(\boldsymbol{o}^{(1)}|\lambda^{(2)})) \tag{7}$$

The two models will be close if the log-likelihoods of sequences are similar, in which case $D(\lambda^{(1)}, \lambda^{(2)})$ is close to 0. Alternatively, we could also use the symmetrical adaptation of D [12], which is given by

$$D_s = \frac{D(\lambda^{(1)}, \lambda^{(2)}) + D(\lambda^{(2)}, \lambda^{(1)})}{2} \tag{8}$$

However, these are both unviable for a deep-learning loss function, as the number of computations is $O(N^2 \sum_{j=1}^{n_o} T_j)$, which will likely result in vanishing and exploding gradients (see Section 4.1.4). The same can be said for information criteria such as Akaike Information Criterion (AIC) and Bayesian Information Criterion (BIC), which require calculation of the log-likelihood of the observation sequences.

5.2 Baum-Welch and Information Criterion

A common approach when the number of states is unknown is to apply the Baum-welch algorithm for different numbers of hidden states, then select the best fitted model using an information criterion such as AIC, BIC or Hannan-Quinn Information Criterion (HQC) [6]. In doing so, we are balancing log-likelihood with the number of parameters in the model. Since we must apply the Baum-Welch algorithm many times, this method is computationally expensive.

5.3 HDP-HMM

A more efficient solution to the unknown number of states is a Bayesian approach which allows for an infinite number of states. One such method is called the Hierarchical Dirichlet Process Hidden Markov Model (HDP-HMM) [13]. We will begin by explaining Dirichlet Processes (Section 5.3.1), then how Hierarchical Dirichlet Processes (HDPs) can be applied to emulate a Hidden Markov Model (Section 5.3.2).

5.3.1 Dirichlet Process

A Dirichlet process G is characterised by a base distribution H and a stickiness parameter α, which we notate as $G \sim DP(H, \alpha)$. Dirichlet processes produce discrete distributions, and the stick-breaking analogy is commonly

[6] For example, we can get a likelihood of 1 if there is a new hidden state for each time step.

used to explain the nature of these distributions [13]. Here, we sample stick-breaking weights $u_k \sim \text{Beta}(1, \alpha)$ for $k \in \mathbb{N}$. We similarly sample infinitely many times from the base distribution, with $\theta_k \sim H, \quad k \in \mathbb{N}$. Now, imagine that we start with a stick of length 1. For each value of $k = 1, 2, ...$, starting with $k = 1$, we break of a proportion of the stick given by u_k and remove it, and the remaining portion of the stick is used for subsequent breaks. The resultant lengths, the stick-breaking weights, are given by $w_k = u_k \prod_{j=1}^{k-1}(1-u_j)$, where $\sum_{k=1}^{\infty} w_k = 1$ and $w_k > 0$ for all $k \in \mathbb{N}$. Therefore, assignment of probability w_k to the sample θ_k produces a probability density function $\sum_{k=1}^{\infty} w_k \delta_{\theta_k}$. This is a distribution with "atoms" at each θ_k, and the weights of each atom (the stick-breaking weights) are influenced by the parameter α; a larger value causes more even distribution of weights, while a smaller value causes fewer weights to dominate others. Some samples from a Dirichlet process can be seen in Figure 6.

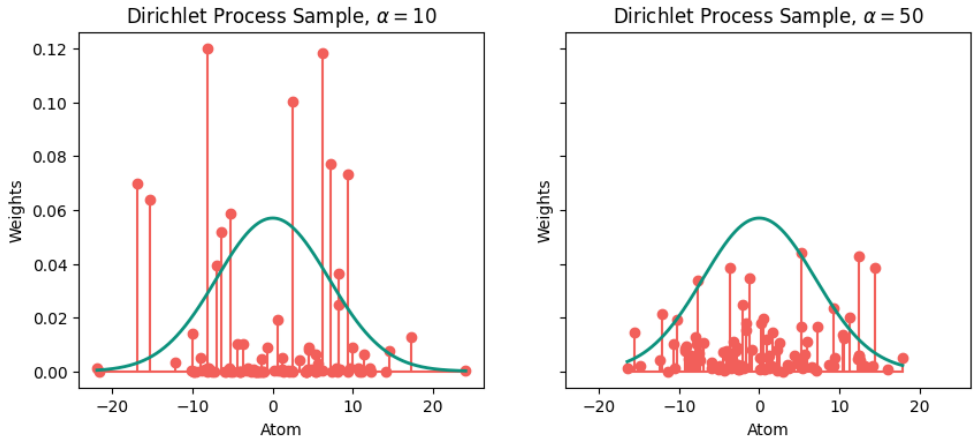


Figure 6: Samples from a dirichlet process with base distribution $N(0, 7^2)$ and $\alpha = 10$ (left) / $\alpha = 50$ (right). A larger value of α leads to higher consistency in atom weights.

The Chinese Restaurant Process is another analogy for Dirichlet processes, this time describing the distribution of one realisation conditioned on all previous realisations (i.e. $\Pr(d_t | d_{t-1}, ..., d_1)$, where $d_i \sim DP(H, \alpha)$). In this analogy, a person walking into a Chinese restaurant represents a new observation, and the table that they sit at represents the discrete output produced from the process. The customer sits at an existing table with probability proportional to the number of people already at the table, and creates a new table with probability proportional to the stickiness parameter. This illustrates the clustering property of Dirichlet Processes [13].

5.3.2 HDPs in HMMs

In a Hidden Markov Model, say we know the state at time t. Then, we select the next state from the transition matrix, which in turn determines the distribution of the next observation. Therefore, our overall emission distribution can be considered as a mixture of distributions from each state, weighted by the transition probability to that state.

In the HDP-HMM, instead of having a set number of states that we can transition to, the state at the next

time step will be distributed according to a Dirichlet Process. Therefore, as described in the Chinese restaurant analogy (see Section 5.3.1), we allow new states to be realised at any time step. Further, the clustering nature of a Dirichlet Process ensures that we are more likely to travel to states which have previously been reached many times from the current state [13], which allows the model to emulate large transition probabilities.

For every current state, we need the possible set of new states to be the same. Therefore, we need each Dirichlet process to share the same atoms across all possibilities for the current state. This is why we draw the base distribution for each "current state" from a Dirichlet distribution as well: it allows us to share new states (although the likelihood of new states is specific to each starting state). This is a hierarchical Dirichlet process, which allows us to generate sequences analogously to an HMM, without specifying the number of states. Therefore, we have arrived at the HDP-HMM, as introduced in [13].

5.4 Discussion of Potential Deep-Learning Approaches

The unknown number of states results in a variable number of outputs required for any deep-learning approach, which prevents the application of the RNN models from Section 4.2. While RNNs are typically able to produce outputs of variable length by aggregating the hidden state at multiple time steps (i.e. a sequence-to-sequence RNN), this enforces order among parameters, as each output is a function of the hidden state, which in turn is a function of the previous hidden state. Therefore, this is not suitable for HMM parameter inference, where there should be no dependency between distributions, for example.

One possible architecture could use pre-trained RNNs as described in Section 4.2. We can first train a new RNN to output a natural number for the number of hidden states, which determines which pre-trained model will be used to derive the parameters from the observation sequences. However, this will be computationally expensive to pre-train RNNs for every possible number of states. The seemingly most promising class of deep-learning models would be attention-based models. This is left as an area for future research.

6 Conclusion

As an alternative to the Baum-Welch algorithm in HMM parameter inference, we investigated the potential for RNN and RNN/Baum-Welch hybrid models. Although we identified problems in all derived loss functions (namely, from state permutations and computational costs), the implementation of the RNN model with JSD as the loss function produced comparable results to the Baum-Welch algorithm. We also surveyed existing approaches when the number of states is unknown: the Baum-Welch algorithm with an information criterion and the HDP-HMM. We again highlighted the computational issue with likelihood-related loss functions for deep-learning approaches, although we outline some potential deep-learning architectures and recommend the use of a transformer-based approach.

References

[1] Bengio, Y., Boulanger-Lewandowski, N. and Pascanu, R. [2012], 'Advances in optimizing recurrent networks'.
URL: *https://arxiv.org/abs/1212.0901*

[2] Briët, J. and Harremoës, P. [2009], 'Properties of classical and quantum Jensen-Shannon divergence', *Phys. Rev. A* **79**, 052311.
URL: *https://link.aps.org/doi/10.1103/PhysRevA.79.052311*

[3] Cover, T. M. and Thomas, J. A. [1991], *Elements of Information Theory*, John Wiley & Sons, New York, NY.

[4] Frigyik, A. B., Kapila, A. and Gupta, M. R. [2010], Introduction to the Dirichlet distribution and related processes.
URL: *https://api.semanticscholar.org/CorpusID:8763665*

[5] Hsu, D., Kakade, S. M. and Zhang, T. [2012], 'A spectral algorithm for learning hidden Markov models'.
URL: *https://arxiv.org/abs/0811.4413*

[6] Jung, S. and Dickson, R. M. [2009], 'Hidden Markov analysis of short single molecule intensity trajectories', *J. Phys. Chem. B* **113**(42), 13886–13890.

[7] Liu, T. and Lemeire, J. [2017], 'Efficient and effective learning of HMMs based on identification of hidden states', *Mathematical Problems in Engineering* **2017**(1), 7318940.
URL: *https://onlinelibrary.wiley.com/doi/abs/10.1155/2017/7318940*

[8] Mor, B., Garhwal, S. and Kumar, A. [2021], 'A systematic review of hidden Markov models and their applications', *Archives of Computational Methods in Engineering* **28**(3), 1429–1448.
URL: *https://doi.org/10.1007/s11831-020-09422-4*

[9] Nielsen, F. [2020], 'On a generalization of the Jensen–Shannon divergence and the Jensen–Shannon centroid', *Entropy* **22**(2), 221.
URL: *http://dx.doi.org/10.3390/e22020221*

[10] Pascanu, R., Mikolov, T. and Bengio, Y. [2013], 'On the difficulty of training recurrent neural networks'.
URL: *https://arxiv.org/abs/1211.5063*

[11] Prince, S. J. [2023], *Understanding Deep Learning*, The MIT Press.
URL: *http://udlbook.com*

[12] Rabiner, L. [1989], 'A tutorial on hidden Markov models and selected applications in speech recognition', *Proceedings of the IEEE* **77**(2), 257–286.

[13] Teh, Y., Jordan, M., Beal, M. and Blei, D. [2006], 'Hierarchical Dirichlet processes', *Machine Learning* pp. 1–30.

[14] Wenzel, F., Snoek, J., Tran, D. and Jenatton, R. [2021], 'Hyperparameter ensembles for robustness and uncertainty quantification'.
URL: *https://arxiv.org/abs/2006.13570*

[15] Yu, D. and Deng, L. [2014], *Automatic Speech Recognition: A Deep Learning Approach*, Springer Publishing Company, Incorporated.

Appendix A All Experimental Results

In Figures 7, 8, 9 and 10, the results for different values of the following variables can be seen:

- models (RNN, Baum-Welch and Baum-Welch Initialisation)
- loss functions (negative log-likelihood, sum of JSDs)
- number of initialisations (1, 5, 10)
- sequence length ranges (10-20, 100-200).

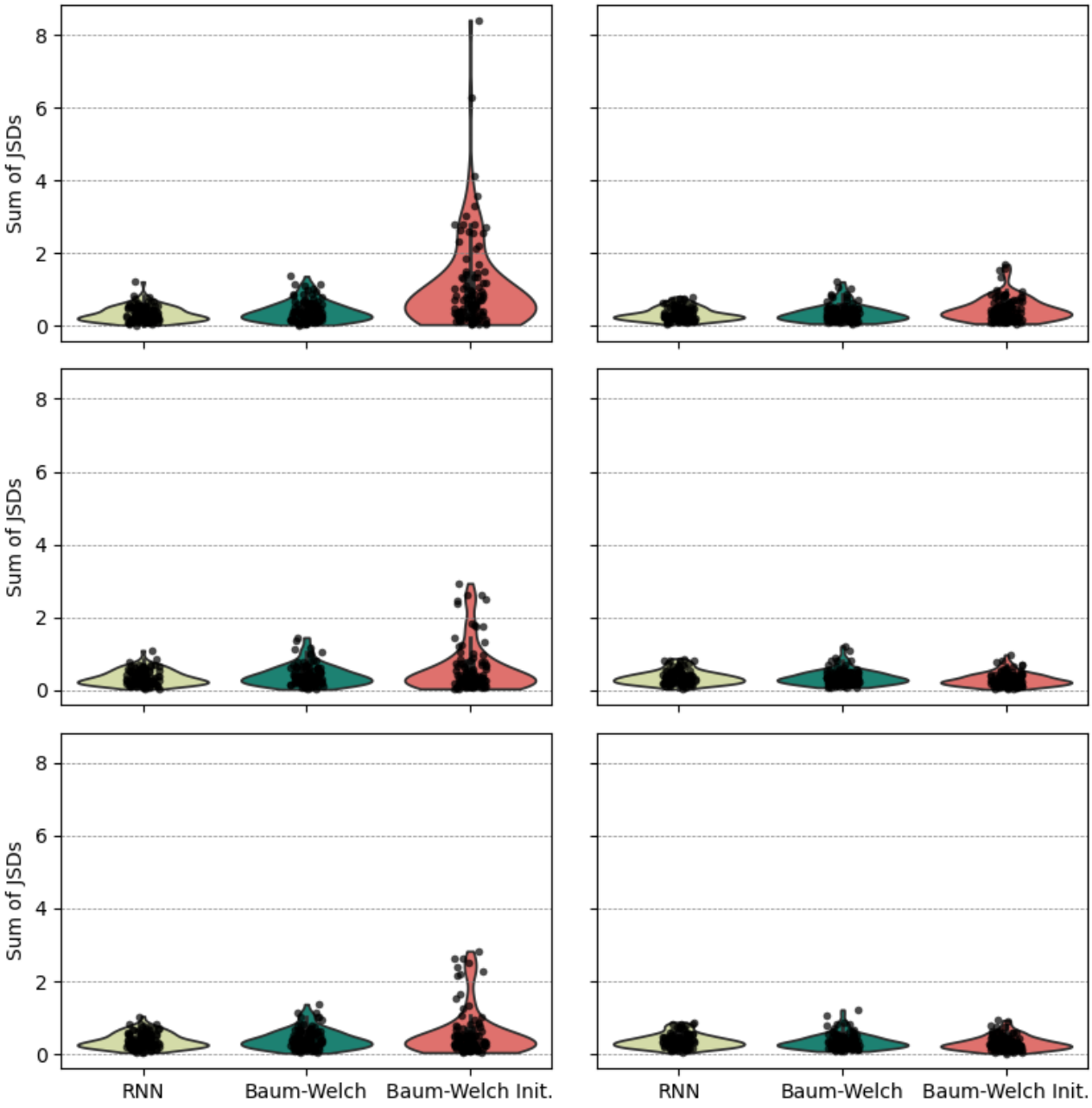


Figure 7: The performance of RNNs trained with JSD (light green), RNN Baum-Welch Initialisation (red) and Baum-Welch (dark green), for **1 sequence**. Smaller loss (concentration near the bottom of each plot) is preferred. **Evaluated with JSD**. Observation sequence length is 10-20 for the left column and 100-200 for the right. Number of initialisations/size of hyperparameter ensemble is 1 for the top row, 5 for the middle and 10 for the bottom.

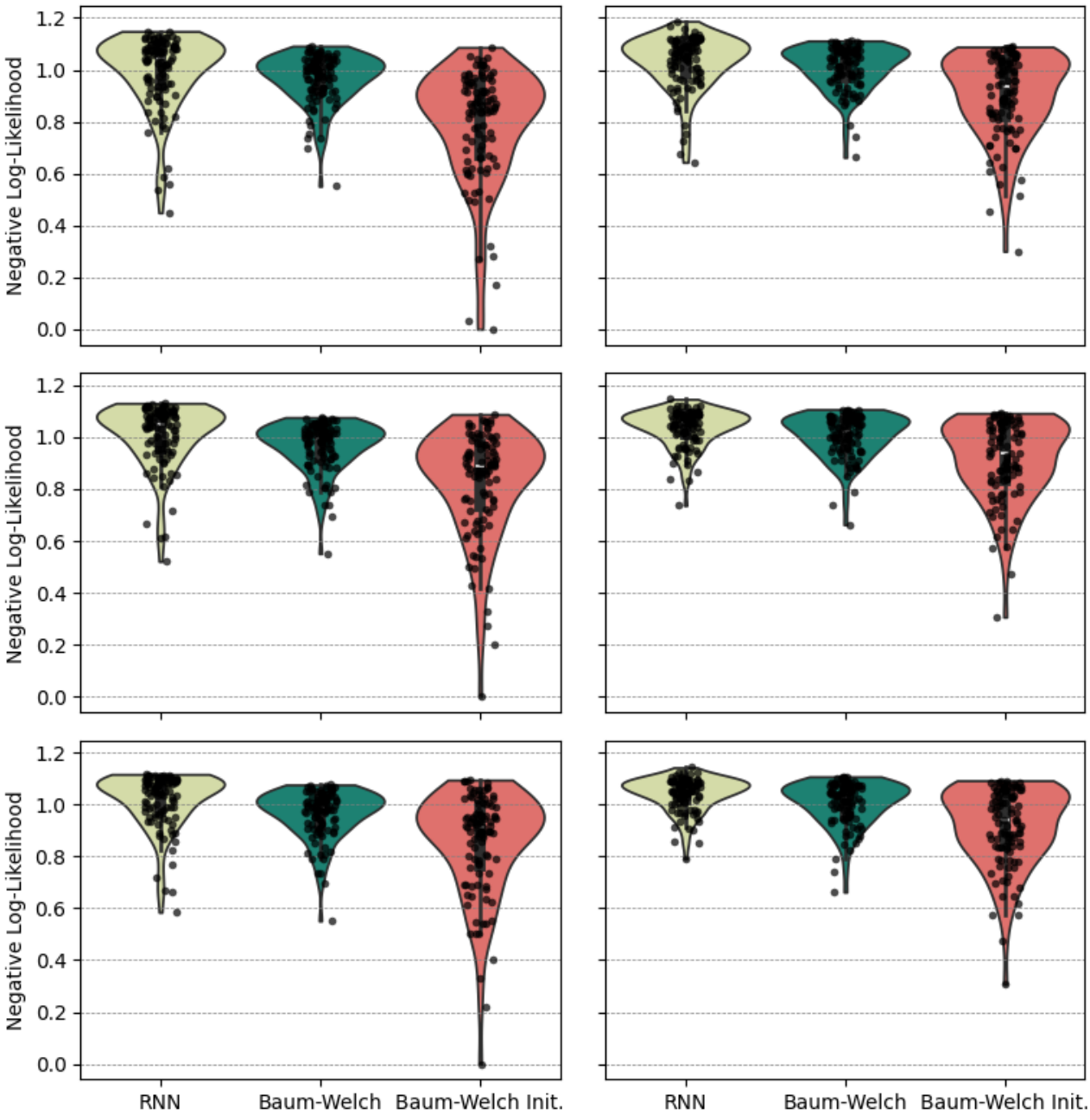


Figure 8: The performance of RNNs trained with JSD (light green), RNN Baum-Welch Initialisation (red) and Baum-Welch (dark green), for **1 sequence**. Smaller loss (concentration near the bottom of each plot) is preferred. **Evaluated with Negative log-likelihood**. Observation sequence length is 10-20 for the left column and 100-200 for the right. Number of initialisations/size of hyperparameter ensemble is 1 for the top row, 5 for the middle and 10 for the bottom.

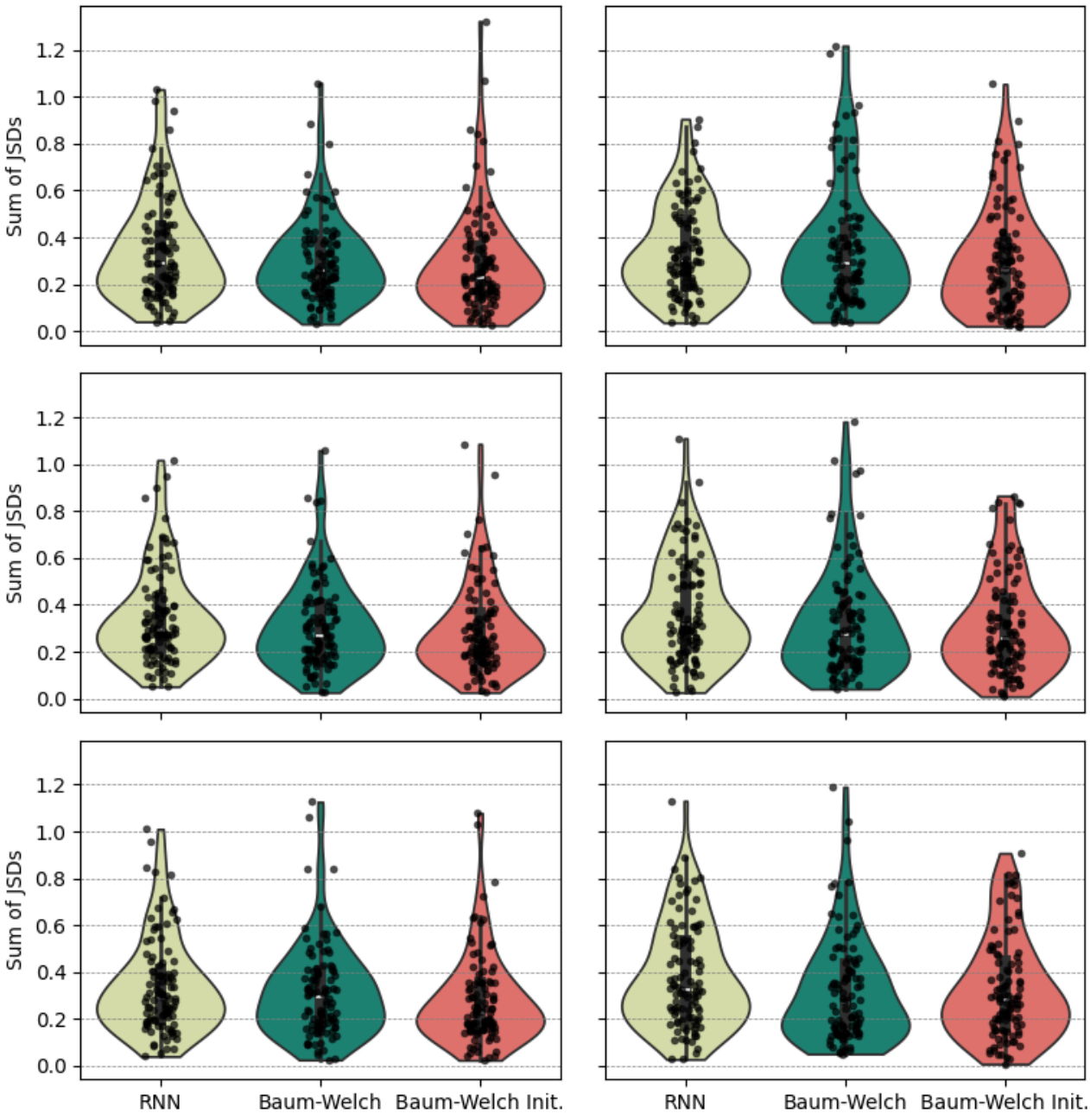


Figure 9: The performance of RNNs trained with JSD (light green), RNN Baum-Welch Initialisation (red) and Baum-Welch (dark green), for **5-10 sequences**. Smaller loss (concentration near the bottom of each plot) is preferred. **Evaluated with JSD**. Observation sequence length is 10-20 for the left column and 100-200 for the right. Number of initialisations/size of hyperparameter ensemble is 1 for the top row, 5 for the middle and 10 for the bottom.

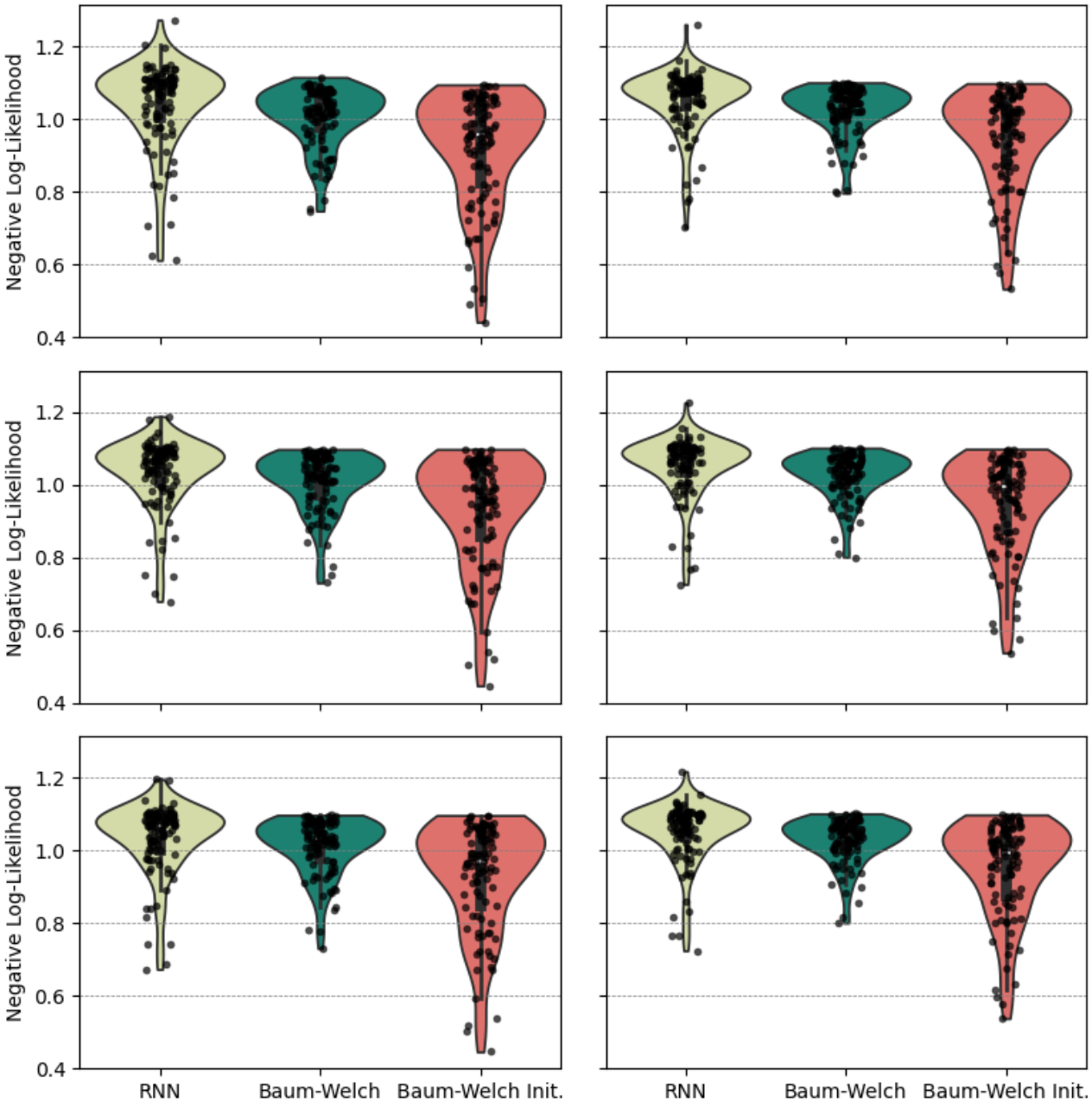


Figure 10: The performance of RNNs trained with JSD (light green), RNN Baum-Welch Initialisation (red) and Baum-Welch (dark green), for **5-10 sequences**. Smaller loss (concentration near the bottom of each plot) is preferred. **Evaluated with Negative log-likelihood**. Observation sequence length is 10-20 for the left column and 100-200 for the right. Number of initialisations/size of hyperparameter ensemble is 1 for the top row, 5 for the middle and 10 for the bottom.

We will now remove the RNN-Baum-Welch hybrid model from Figures 7, 8, 9 and 10 to allow more suitable axes and easier comparison between the RNN and Baum-Welch. The resultant plots are in Figures 11, 12, 13 and 14.

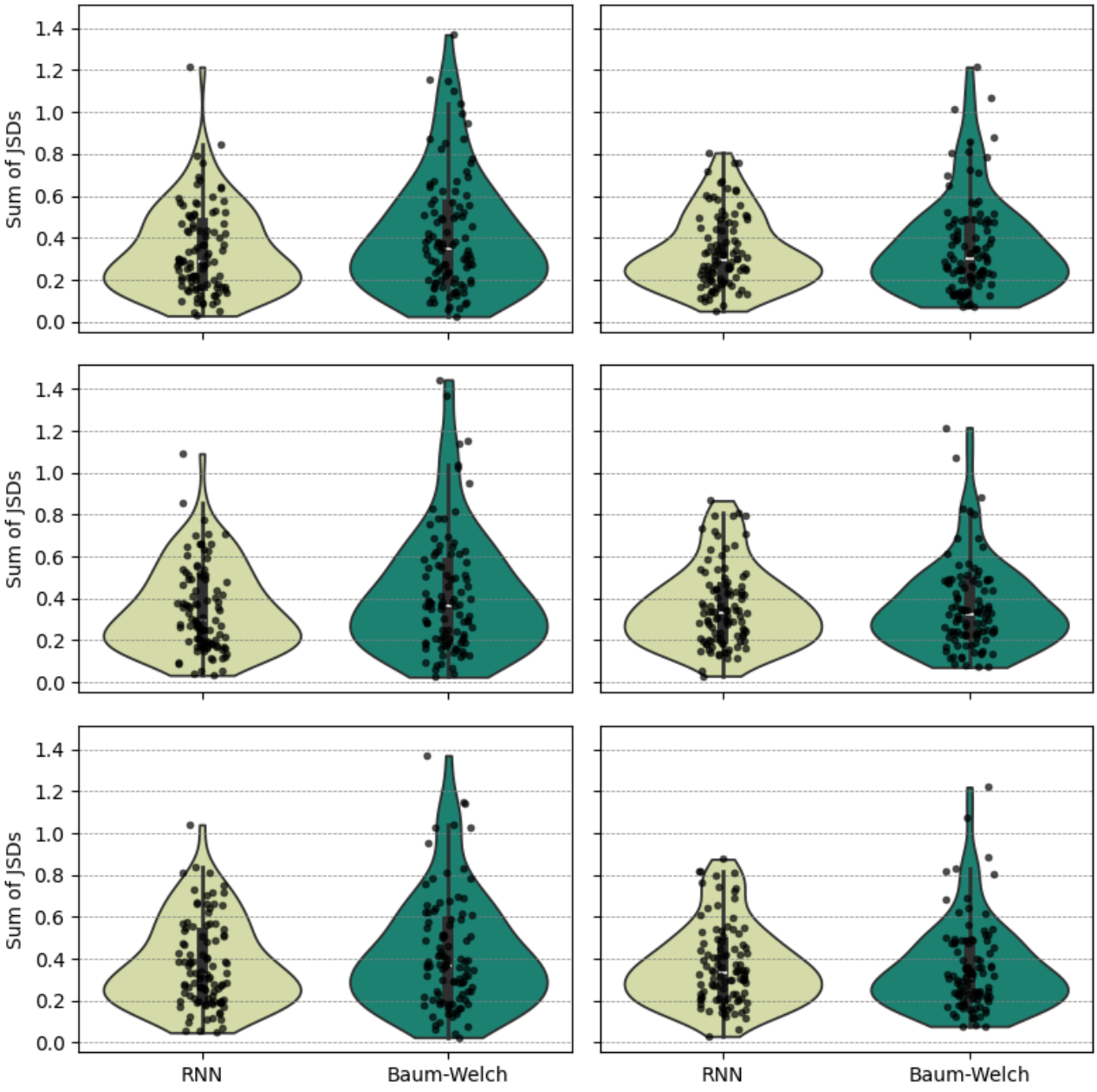


Figure 11: The performance of RNNs trained with JSD (light green), and Baum-Welch (dark green), for **1 sequence**. Smaller loss (concentration near the bottom of each plot) is preferred. **Evaluated with JSD**. Observation sequence length is 10-20 for the left column and 100-200 for the right. Number of initialisations/size of hyperparameter ensemble is 1 for the top row, 5 for the middle and 10 for the bottom.

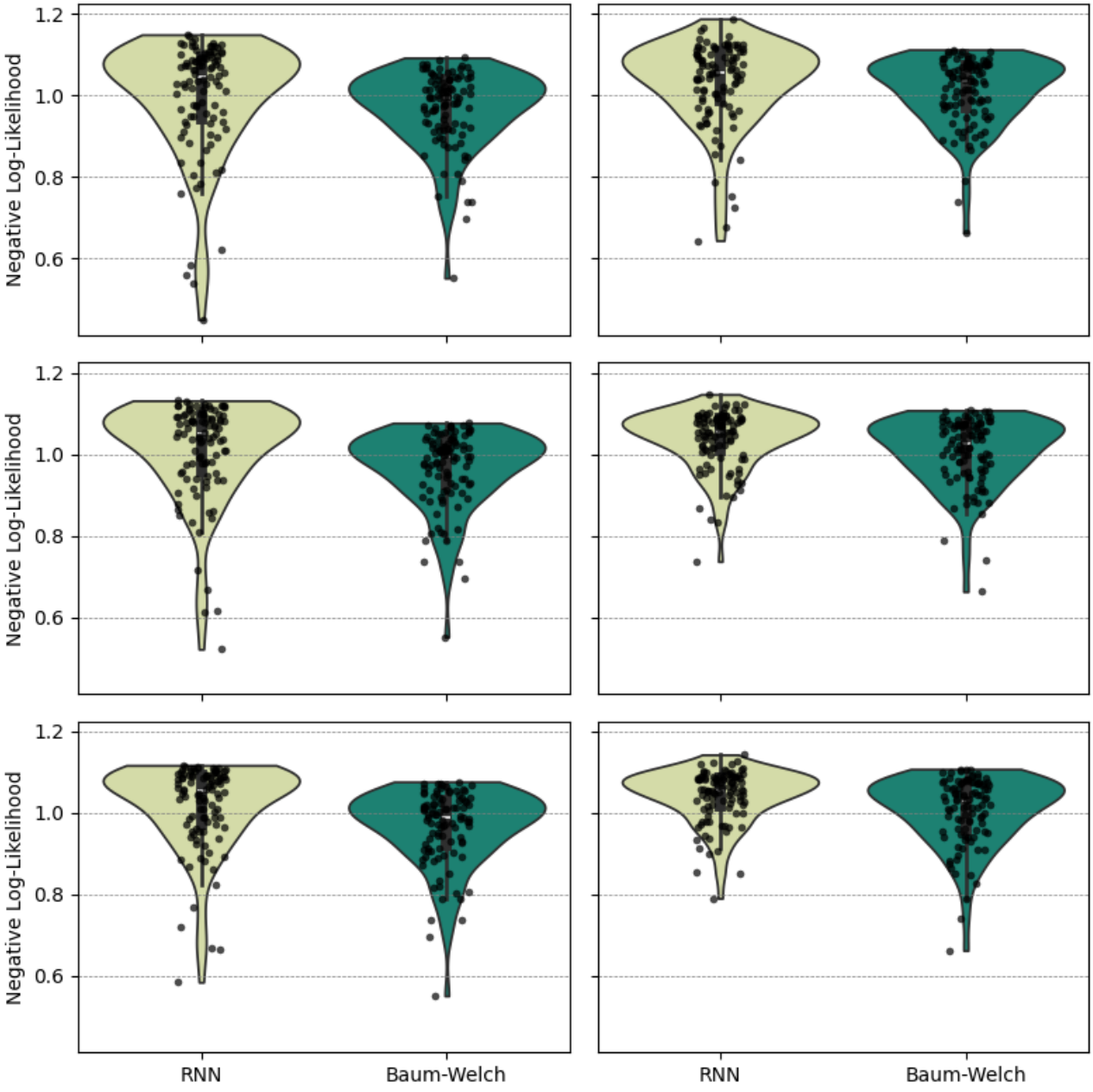


Figure 12: The performance of RNNs trained with JSD (light green), and Baum-Welch (dark green), for **1 sequence**. Smaller loss (concentration near the bottom of each plot) is preferred. **Evaluated with Negative log-likelihood**. Observation sequence length is 10-20 for the left column and 100-200 for the right. Number of initialisations/size of hyperparameter ensemble is 1 for the top row, 5 for the middle and 10 for the bottom.

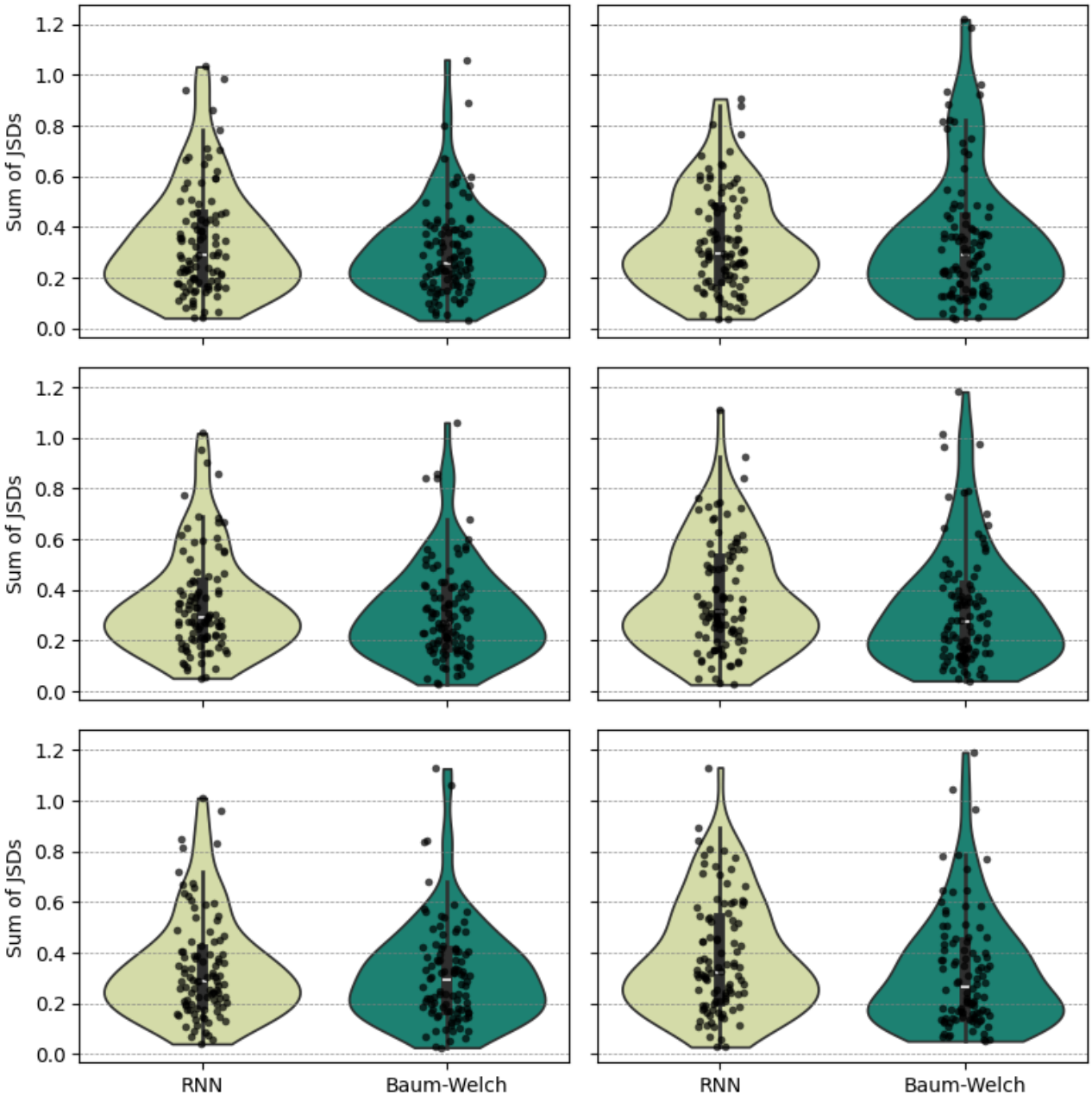


Figure 13: The performance of RNNs trained with JSD (light green), and Baum-Welch (dark green), for **5-10 sequences**. Smaller loss (concentration near the bottom of each plot) is preferred. **Evaluated with JSD**. Observation sequence length is 10-20 for the left column and 100-200 for the right. Number of initialisations/size of hyperparameter ensemble is 1 for the top row, 5 for the middle and 10 for the bottom.

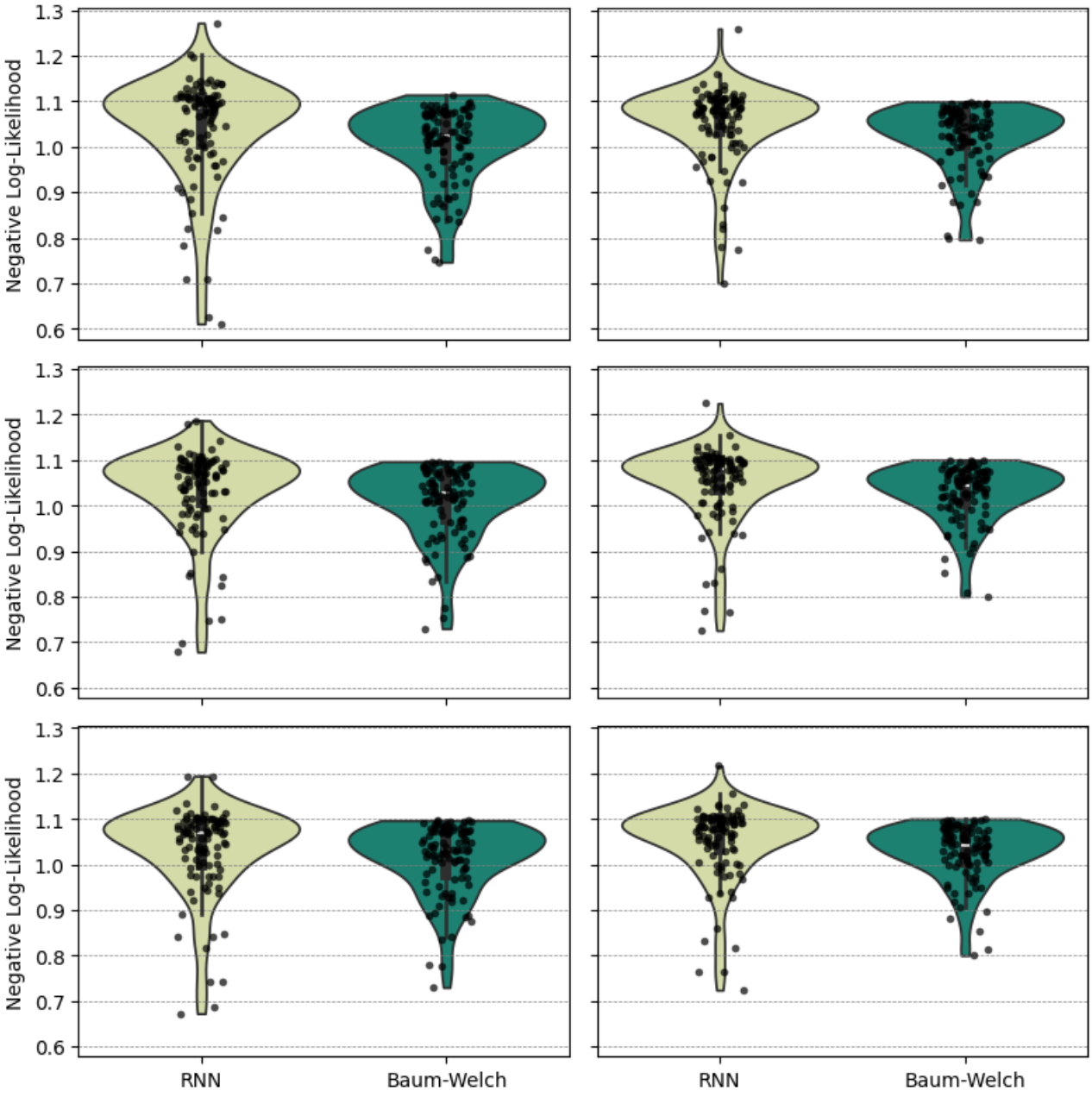


Figure 14: The performance of RNNs trained with JSD (light green), and Baum-Welch (dark green), for **5-10 sequences**. Smaller loss (concentration near the bottom of each plot) is preferred. **Evaluated with Negative log-likelihood**. Observation sequence length is 10-20 for the left column and 100-200 for the right. Number of initialisations/size of hyperparameter ensemble is 1 for the top row, 5 for the middle and 10 for the bottom.