Particle Filter Recurrent Neural Networks

Xiao Ma* **Peter Karkus*** **David Hsu** **Wee Sun Lee**

School of Computing, National University of Singapore

{xiao-ma, karkus, dyhsu, leews}@comp.nus.edu.sg

Abstract

Recurrent neural networks (RNNs) have been extraordinarily successful for prediction with sequential data. To tackle highly variable and noisy real-world data, we introduce *Particle Filter Recurrent Neural Networks* (PF-RNNs), a new RNN family that explicitly models uncertainty in its internal structure: while an RNN relies on a long, deterministic latent state vector, a PF-RNN maintains a latent *state distribution*, approximated as a set of particles. For effective learning, we provide a fully differentiable particle filter algorithm that updates the PF-RNN latent state distribution according to the Bayes rule. Experiments demonstrate that the proposed PF-RNNs outperform the corresponding standard gated RNNs on a synthetic robot localization dataset and 10 real-world sequence prediction datasets for text classification, stock price prediction, etc.

1 Introduction

Prediction with sequential data is a long-standing challenge in machine learning. It has many applications, e.g., object tracking [1], robot localization [36], speech recognition [38], and decision making under uncertainty [35]. For effective prediction, predictors require "memory", which summarizes and tracks information in the input sequence. The memory state is generally not observable, hence the need for a *belief*, i.e., a posterior state distribution that captures the sufficient statistic of the input for making predictions. Modeling the belief manually is often difficult. Consider the task of classifying news text—treated as a sequence of words—into categories, such as politics, education, economy, etc. It is difficult to handcraft the belief representation and dynamics for accurate classification.

State-of-the-art sequence predictors often use recurrent neural networks (RNNs), which *learn* a vector h of deterministic time-dependent latent variables as an approximation to the belief. Real-world data, however, are highly variable and noisy. To cope with the complexity of uncertain real-world data and achieve better belief approximation, one could increase the length of latent vector h, thus increasing the number of network parameters and the data required for training the network.

We introduce *Particle Filter Recurrent Neural Networks* (PF-RNNs), a new family of RNNs, which seek to improve belief approximation without lengthening the latent vector h, thus reducing the data required for learning. A PF-RNN approximates the belief as a set of weighted latent vectors $\{h^1, h^2, \ldots\}$ sampled from the same distribution, just as *particle filtering* [9] does. Particle filtering is a model-based belief tracking algorithm. It approximates the belief as a set of sampled states that typically have well-understood meaning. Like standard RNNs, PF-RNNs follow a model-free approach: PF-RNNs' latent vectors are learned distributed representations, which are not necessarily interpretable. PF-RNNs borrow from particle filtering the idea of approximating the belief as a set of weighted particles, and combine it with the powerful approximation capacity of RNNs. The approximate representation is trained from data to optimize the prediction performance. For effective training with gradient methods, we employ a fully differentiable particle filter algorithm that maintains the latent belief. See Fig. 1 for a comparison of RNN and PF-RNN.

*equal contribution

Preprint. Under review.

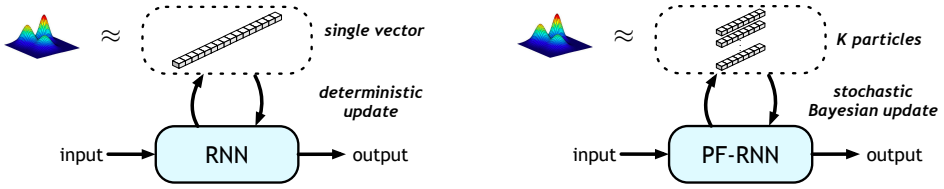

Figure 1: **A comparison of RNN and PF-RNN**. An RNN approximates the belief as a long latent vector and updates it with a deterministic nonlinear function. A PF-RNN approximates the belief as a set of weighted particles and updates them with the stochastic particle filtering algorithm.

We apply the underlying idea of PF-RNN to gated RNNs, which have shown strong performance in many sequence prediction tasks. Specifically, we propose PF-LSTM and PF-GRU, the particle filter extensions of Long Short Term Memory (LSTM) [13] and Gated Recurrent Unit (GRU) [7]. PF-LSTM and PF-GRU serve as drop-in replacements for LSTM and GRU, respectively. They aim to learn a better belief representation from the same data, though at a greater computational cost.

We evaluated PF-LSTM and PF-GRU on 11 data sets: 1 synthetic dataset for systematic understanding and 10 real-world datasets for performance comparison. The experiments show that our PF-RNNs outperform the corresponding standard RNNs with a comparable number of parameters. Further, the PF-RNNs achieve the best results on almost all datasets when there is no restriction on the number of model parameters used.

2 Related Work

There are two general approaches to prediction with sequential data: model-based and model-free. The model-based approach includes, e.g., the well-known latent Markov models (HMMs) and the dynamic Bayesian networks (DBNs) [29]. They rely on handcrafted state representations with well-defined semantics, e.g., phonemes in speech recognition. Given a model, one may perform belief tracking according to the Bayes rule. The main difficulty here is that the state space and consequently the computational complexity of belief tracking grow exponentially with the number of state dimensions. To cope with this difficulty, particle filters represent the belief as a set of sampled states and perform approximate inference. Alternatively, the model-free approach, such as RNNs, approximates the belief as a latent state vector, learned directly from data, and updates it through a deterministic nonlinear function, also learned from data. RNNs have produced state-of-the-art results in many sequence prediction tasks, e.g., speech recognition [38].

The proposed PF-RNNs build upon RNNs and combine their powerful data-driven approximation capabilities with the sample-based belief representation and approximate Bayesian inference used in particle filters. Related sample-based methods have been applied to generative models. Importance sampling is used to improve variational auto-encoders [3]. This is extended to sequence generation [23, 27, 30] and to reinforcement learning [15]. Unlike the earlier works that focus on generation, we combine RNNs and particle filtering for sequence prediction. PF-RNNs are trained *discriminatively*, instead of generatively, with the target loss function on the model output. As a result, PF-RNN training prioritizes target prediction over data generation which may be irrelevant to the prediction task.

PF-RNNs exploit the general idea of embedding algorithmic priors, in this case, filtering algorithms, in neural networks and train them discriminatively [12, 17, 20, 18, 21]. Earlier work embeds a particle filter in an RNN for learning belief tracking, but follows a model-based approach and relies on handcrafted belief representation [18, 21]. PF-RNNs retain the model-free nature of RNNs and exploit their powerful approximation capabilities to learn belief representation directly from data. Other work explicitly addresses belief representation learning with RNNs [10, 11, 28]; however, they do not involve Bayesian belief update or particle filtering.

3 Particle Filter Recurrent Neural Networks (PF-RNNs)

3.1 Overview

The general sequence prediction problem is to predict an output sequence, given an input sequence. In this paper, we focus on predicting the output y_t at time t, given the input history $x_1, x_2, \ldots, x_t$.

Standard RNNs handle sequence prediction by maintaining a deterministic latent state h_t that captures the sufficient statistic of the input history, and updating h_t sequentially given new inputs. Specifically, RNNs update h_t with a deterministic nonlinear function learned from data. The predicted output $\hat{y}_t$ is another nonlinear function of the latent state h_t, also learned from data.

To handle highly variable, noisy real-world data, one key idea of PF-RNN is to capture the sufficient statistic of the input history in the latent *belief* $b(h_t)$ by forming multiple hypotheses over h_t. Specifically, PF-RNNs approximate $b(h_t)$ by a set of K weighted particles $\{(h_t^i, w_t^i)\}_{i=1}^K$, for latent state h_t^i and weight w_t^i; the particle filtering algorithm is used to update the particles according to the Bayes rule. The Bayesian treatment of the latent belief naturally captures the stochastic nature of real-world data. Further, all particles share the same parameters in a PF-RNN. The number of particles thus does not affect the number of PF-RNN network parameters. Given a fixed training data set, we expect that increasing the number of particles improves the belief approximation and leads to better learning performance, but at the cost of greater computational complexity.

Similar to RNNs, we used learned functions for updating latent states and predict the output y_t based on the averaged particle: $\hat{y}_t = f_{\text{out}}(\bar{h}_t)$, where $\bar{h}_t = \sum_{i=1}^K w_t^i h_t^i$ and f_{out} is a task-dependent prediction function. For example, in a robot localization task, f_{out} maps $\bar{h}_t$ to a robot position. In a classification task, f_{out} maps $\hat{y}_t$ to a vector of class probabilities.

For learning, we embed a particle filter algorithm in the network architecture and train the network discriminatively end-to-end. Discriminative training aims for the best prediction performance within the architectural constraints imposed by the particle filter algorithm. We explore two loss functions for learning: one minimizes the error of prediction with the mean particle and the other maximizes the log-likelihood of the target y_t given the full particle distribution. Combining the two loss functions gives the best performance empirically.

3.2 Particle Filter Extension to RNNs

Extending an RNN to the corresponding PF-RNN requires the latent particle belief representation $\{(h_t^i, w_t^i)\}_{i=1}^K$ and the associated belief update function. The new belief is a posterior distribution conditioned on the previous belief and the new input. We model the PF-RNN's latent dynamics as a controlled system, with the uncontrolled system as a special case. The standard belief update consists of two steps: the transition update $\tilde{b}_t = f_{\text{tr}}(b_{t-1}, u_t)$ for control u_t and the observation update $b_t = f_{\text{obs}}(\tilde{b}_t, o_t)$ for observation o_t. However, u_t and o_t are not separated a priori in general sequence prediction problems. PF-RNNs use input x_t in both f_{tr} and f_{obs}, and learn to extract latent control $u_t(x_t)$ and observation $o_t(x_t)$ from x_t through task-oriented discriminative training. This approach is more flexible and provides a richer function approximation class.

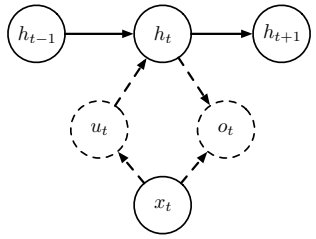

Figure 2: **The graphical model for PF-RNN belief update**. The dashed circles indicate variables not explicitly defined in our models.

Stochastic memory update. We apply a learned transition function f_{tr} to each particle state h_t^i:

$$h_t^i = f_{\text{tr}}(h_{t-1}^i, u_t(x_t), \xi_t^i), \quad \xi_t^i \sim p(\xi_t^i | h_{t-1}^i, u_t(x_t)) \tag{1}$$

where x_t is the input and ξ_t^i is a learned noise term. We assume $p(\xi_t^i | h_{t-1}^i, u_t(x_t))$ to be a Gaussian distribution and use the *reparameterization trick* [22] which uses random inputs to the function to simulate the transition in a differentiable manner. From the RNN perspective, ξ_t^i captures the stochasticity in the latent dynamics. From the particle filtering perspective, ξ_t^i increases particle diversity, relieving the issue of particle depletion after resampling.

Particle weight update. We update the weight w_t^i of the particle h_t^i recursively in a Bayesian manner, using the likelihood $p(o_t | h_t^i)$. Instead of modeling $p(o_t | h_t^i)$ as a generative distribution, we approximate it directly as a learned function $f_{\text{obs}}(o_t(x_t), h_t^i)$ with $o_t(x_t)$ and h_t^i as inputs, and have

$$w_t^i = \eta f_{\text{obs}}(o_t(x_t), h_t^i) w_{t-1}^i, \tag{2}$$

where η is a normalization factor. Our observation model f_{obs}, coupled with discriminative training, significantly improves the performance of PF-RNNs in practice. The generative distribution $p(o_t | h_t^i)$ is parameterized for generating o_t and may contain irrelevant features for the actual objective of predicting y_t. In contrast, f_{obs} skips generative modeling and learns only features useful for predicting y_t, as PF-RNNs are trained discriminatively to optimize prediction accuracy.

Soft resampling. In particle filter algorithms, resampling is required to avoid particle degeneracy, i.e., most particles having near-zero weights. Resampling constructs a new particle set $\{(h_t'^j, w_t'^j)\}_{j=1}^K$ with constant weights $w_t'^j = 1/K$. Each new particle $h_t'^j$ takes the value of an *ancestor* particle, $h_t^{a^j}$, where the ancestor index a^j is sampled from a multinomial distribution defined by the original particle weights, $a^j \sim p$, where p is a multinomial distribution with $p(i) = w_t^i$.

However, resampling is not differentiable, which prevents the use of backpropagation for training PF-RNNs. To make our latent belief update differentiable, we use *soft resampling* [21]. Instead of resampling particles according to p, we resample from q, a mixture of p and a uniform distribution: $q(i) = \alpha w_t^i + (1-\alpha)(1/K)$, for $\alpha \in (0, 1]$. The new weights are computed according the importance sampling formula, which leads to an unbiased estimate of the belief:

$$w_t'^j = \frac{p(i = a^j)}{q(i = a^j)} = \frac{w_t^{a^j}}{\alpha w_t^{a^j} + (1-\alpha)(1/K)} \tag{3}$$

Soft resampling provides non-zero gradients when $\alpha > 0$. We use $\alpha = 0.5$ in our experiments.

3.3 Model Learning

One natural objective is to minimize the total prediction loss over all training sequences. For each training sequence, the prediction loss is

$$L_{\text{pred}}(\theta) = \sum_{t \in \mathcal{O}} \ell(y_t, \hat{y}_t, \theta) \tag{4}$$

where θ represents the predictor parameters, $\mathcal{O}$ is the set of time indices with outputs, and ℓ is a task-dependent loss function measuring the difference between the target output y_t and the predicted output $\hat{y}_t$. Predictions in PF-RNN are made using the mean particle, $\hat{y}_t = f_{\text{out}}(\bar{h}_t)$, where f_{out} is a learned function, and $\bar{h}_t = \sum_{i=1}^K w_t^i h_t^i$ is the weighted mean of the PF-RNN particles. In our regression experiments, $\hat{y}_t$ is a real value and ℓ is the squared loss. In our classification experiments, $\hat{y}_t$ is a multinomial distribution over classes and ℓ is the cross-entropy loss.

The PF-RNN prediction $\hat{y}_t$ is actually randomized, as its value depends on random variables used for stochastic memory updates and soft-resampling; hence, a reasonable objective would be to minimize $\mathrm{E}[L_{\text{pred}}(\theta)]$. In our algorithm, we optimize $L_{\text{pred}}(\theta)$ instead of $\mathrm{E}[L_{\text{pred}}(\theta)]$. We discuss the effect of this approximation in Appendix A.

Another possible learning objective applies stronger model assumptions. To maximize the likelihood, we optimize a sampled version of an *evidence lower bound* (ELBO) of $p(y_t|x_{1:t}, \theta)$,

$$L_{\text{ELBO}}(\theta) = -\sum_{t \in \mathcal{O}} \log \frac{1}{K} \sum_{i=1}^K p(y_t|\tau_{1:t}^i, x_{1:t}, \theta). \tag{5}$$

where $\tau_{1:t}^i$ is a history chain for particle i, consisting of its ancestor indices during particle resampling and the random numbers used in stochastic memory updates[2]. The derivation of the ELBO follows from [3, 23, 27, 30], and it is provided in Appendix B.

The $p(y_t|\tau_{1:t}^i, x_{1:t}, \theta)$ terms in $L_{\text{ELBO}}(\theta)$ are computed using appropriate probabilistic models. We apply f_{out} to each particle at each time step, generating outputs $\hat{y}_t^i = f_{\text{out}}(h_t^i)$. Then, for classification $p(y_t|\tau_{1:t}^i, x_{1:t}, \theta) = \text{CrossEntropy}(y_t, \hat{y}_t^i)$, which follows from the multinomial model; and for regression $p(y_t|\tau_{1:t}^i, x_{1:t}, \theta) = \exp(-||y_t - \hat{y}_t^i||)$, which follows from a unit variance Gaussian model.

Intuitively, L_{pred} encourages PF-RNN to make good predictions using the mean particle, while L_{ELBO} encourages the model to learn a more meaningful belief distribution. They make different structure assumptions that result in different gradient flows. Empirically, we find that combining the two learning objectives works well: $L(\theta) = L_{\text{pred}}(\theta) + \beta L_{\text{ELBO}}(\theta)$, where β is a weight parameter. We use $\beta = 1.0$ in the experiments.

3.4 PF-LSTMs and PF-GRUs

We now apply the PF-RNN to the two most popular RNN architectures, LSTM and GRU.

[2]We have negated the ELBO to make it consistent with loss minimization.

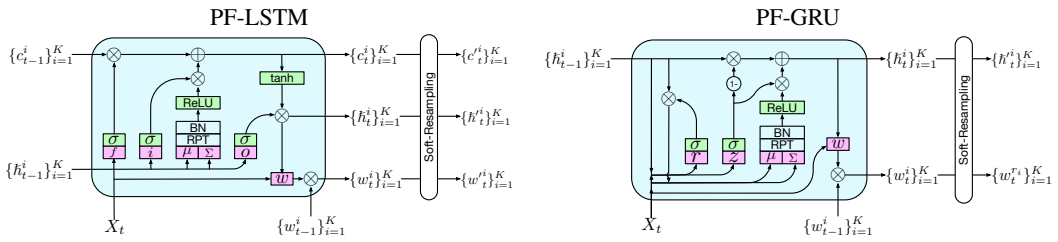

Figure 3: **PF-LSTM and PF-GRU network architecture**. Notation (1) green box: activation (2) pink box: learned function (3) *RPT*: reparameterization trick (4) *BN*: batch normalization.

For standard LSTM, the memory state h_t consists of cell state c_t and hidden state $\hbar_t$. The memory update is a deterministic mapping controlled by input gate i_t, forget gate f_t, and output gate o_t:

$$c_t = f_t \circ c_{t-1} + i_t \circ \tanh(\tilde{c}_t), \quad \tilde{c}_t = W_c[\hbar_{t-1}, x_t] + b_c, \quad \hbar_t = o_t \circ \tanh(c_t), \tag{6}$$

where x_t is the current input and $\circ$ is the element-wise product. For PF-LSTM, the memory state consists of a set of weighted particles $\{(\hbar_t^j, c_t^j, w_t^j)\}_{j=1}^K$. To help PF-LSTM track the latent particle belief effectively over a long history of data, we make two changes to the memory update equations. One is to add stochasticity:

$$\tilde{c}_t^j = W_c[\hbar_{t-1}^j, x_t] + b_c + \xi_t^j, \quad \xi_t^j \sim \mathcal{N}(0, \Sigma_t^j), \quad \Sigma_t^j = W_\Sigma[\hbar_{t-1}^j, x_t] + b_\Sigma, \tag{7}$$

for $j = 1, 2, \ldots, K$. The motivation for stochastic memory updates is discussed in Sect. 3.2. The other change, inspired by LiGRU [33], is to replace the hyperbolic tangent activation of LSTM by ReLU activation and batch normalization [16]:

$$c_t^j = f_t^j \circ c_{t-1}^j + i_t^j \circ \text{ReLU}(\text{BN}(\tilde{c}_t^j)), \tag{8}$$

for $j = 1, 2, \ldots, K$. Recurrent networks are usually trained with truncated Back-Propagation-Through-Time (BPTT) [37]. Truncation affects the training of PF-RNNs, more than that of RNNs, as PF-RNNs maintain the latent belief explicitly and may need a long sequence of inputs in order to approximate the belief well. As shown in [33], ReLU activation, combined with batch normalization, has better numerical properties for backpropagation through many time steps, and thus it allows us to use longer sequence length with truncated BPTT. The update equation for $\hbar_t^j$ remains the same as that for LSTM. After updating the particles and their weights, we perform soft resampling to get a new set of particles (Section 3.2). The PF-LSTM architecture is shown in Fig. 3 (left side).

The PF-GRU model can be constructed similarly (right side of Fig. 3). The details are available in Appendix C. The same idea can be easily applied to other gated recurrent neural networks as well.

4 Experiments

We evaluate PF-RNNs, specifically, PF-LSTM and PF-GRU, on a synthetic 2D robot localization task and 10 sequence prediction datasets from various domains. We compare PF-RNNs with the corresponding RNNs. First, we use a fixed latent state size for all models. The sizes are chosen so that the PF-RNN and the RNN have roughly the same number of trainable parameters. Next, we search over the latent state size for all models independently and compare the best results achieved. We also perform an ablation study to understand the effect of individual components of PF-RNNs. Implementation details are in Appendix D. We will release the code upon publication of the paper.

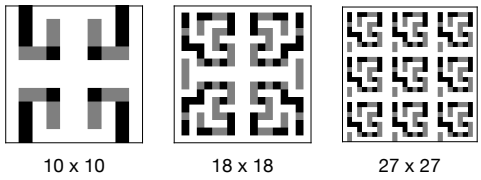

Figure 4: **Localization in a symmetric maze**. Each maze is an $N \times N$ grid, with black and gray obstacles. Black obstacles serve also as landmarks.

4.1 Robot Localization

We first evaluate PF-RNNs in a simple synthetic domain to gain some insights on the approach. The task is to localize a robot on a 2D map (Fig. 4), given distance measurements to a set of landmarks. In each environment, half of the obstructions have landmarks placed on their corners.

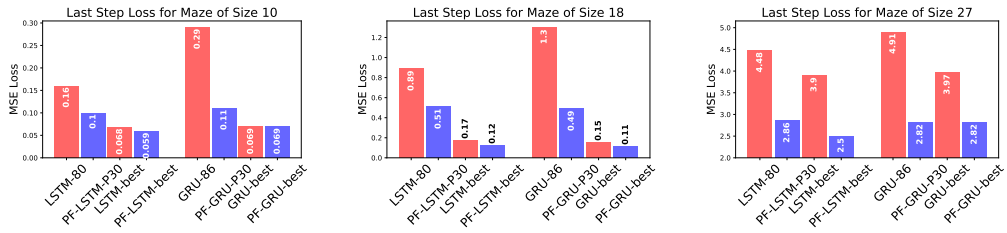

Figure 5: **Performance comparison of the last-step prediction loss in robot localization**. LSTM-80 and GRU-86 indicate LSTM and GRU with latent state size of 80 and 86, respectively. PF-LSTM-P30 and PF-GRU-P30 indicate PF-LSTM and PF-GRU with 30 particles and latent state size of 64. These parameters are chosen so that the RNN and the corresponding PF-RNN have roughly the same number of trainable parameters. LSTM-best, GRU-best, PF-LSTM-best, and PF-GRU-best indicate the best-performing model after a search over the latent state size and the number of particles.

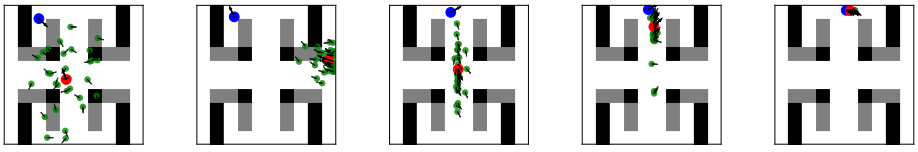

Figure 6: **Visualization of PF-LSTM latent particles**. Each figure shows the true robot pose (blue), predicted pose (red), and 30 predicted poses according to the latent particles (green).

The maps are designed to be highly symmetric, thus maintaining a belief is essential for localization. The robot starts at a random location. In each step, it moves forward a distance $d \sim \mathcal{N}(0.2, 0.05)$ or chooses a new heading direction when about to hit a wall. The input x_t consists of the last action u_{t-1} and current observation o_t, which is a set of noisy distance measurements to the 5 closest landmarks. The observation noise follows the Gaussian model $\mathcal{N}(0, 1)$. The model takes x_t as input and has no prior information on which components represent u_t and o_t. This information must be extracted from x_t through learning. The task is to predict the robot pose at each time t, 2D coordinates and heading, given the history of inputs $\{x_i\}_{i=1}^t$.

We train models on a set of $10,000$ trajectories. We evaluate and test on another $1,000$ and $2,000$ trajectories, respectively. The network architecture is the same for all models. Map features are extracted from the map using convolutional layers, two for the smaller maps and 5 for the largest map. Control and observation features are extracted from the inputs by two fully connected layers. The features are concatenated and input to the recurrent cell, PF-LSTM/PF-GRU or LSTM/GRU. The output of the recurrent cell is mapped to the pose prediction by a fully connected layer. The training loss is the Mean Square Error (MSE) between the predicted and ground truth pose, summed along the trajectories. The last step MSE is used as the evaluation metric. For all models, we perform a standard grid search over training parameters: learning rate, batch size, and gradient clipping value.

We compare PF-LSTM and PF-GRU with LSTM and GRU (Fig. 5). Results show that PF-RNNs consistently outperform the corresponding RNNs. The performance benefits become more pronounced, as the size of the maze grows, resulting in increased difficulty of belief tracking.

Fig. 6 visualizes the particle belief progression in a trained PF-LSTM. Additional examples are in Appendix E.3. PF-LSTM works similarly to a standard particle filtering algorithm and demonstrates a reasonable belief distribution. The particle predictions of the robot pose are initially spread out ($t = 0$). As inputs accumulate, they begin to converge on some features, *e.g.*, horizontal position ($t = 22$); and eventually they converge to the true pose ($t = 26$). The depicted example also demonstrates an interesting property of PF-LSTM. Initially, particle predictions converge towards a wrong, distant pose ($t = 6$), because of the ambiguous observations in the symmetric environment. However, particle predictions spread out again ($t = 22$), and converge to the true pose ($t = 26$). This would be unusual for a standard particle filtering algorithm under the true robot dynamics: once particles converge to a wrong pose, it would be difficult to recover [36]. PF-LSTM succeeds here,

Table 1: Performance comparison on 10 sequence prediction tasks.

	Regression				Classification					
	NASDAQ	AEP	AIR	PM	LPA	AREM	GAS	MR	R52	UID
LSTM	37.33	6.53	18.34	26.14	91.7	99.1	76.6	73.1	81.1	93.2
PF-LSTM	4.65	4.57	**13.12**	21.23	**100.0**	99.1	89.9	78.3	90.1	95.3
LSTM-best	2.53	6.53	17.69	26.14	98.3	100.0	81.2	73.1	81.1	99.1
PF-LSTM-best	**1.82**	**3.72**	**13.12**	**19.04**	**100.0**	**100.0**	**94.1**	**82.2**	**91.3**	**99.6**
GRU	4.93	6.57	17.70	23.59	97.8	98.4	76.8	75.1	84.2	96.1
PF-GRU	**1.33**	5.33	19.32	20.77	98.1	99.1	**83.3**	76.2	87.2	94.1
GRU-best	4.93	5.61	**14.78**	23.59	97.8	**100.0**	80.0	75.2	84.2	99.0
PF-GRU-best	**1.33**	**3.73**	18.18	**20.77**	**99.2**	**100.0**	**83.3**	**79.6**	**89.1**	**99.5**
SOTA	**0.33**[32]	–	–	–	91.3[19]	94.4[31]	80.9[14]	**83.1**[39]	**93.8**[24]	–

because its latent representation and dynamics are learned from data. Each particle in a PF-LSTM independently aggregates information from the input history, and its dynamics is not constrained explicitly by the robot dynamics.

4.2 General Sequence Prediction Tasks

Comparison with RNNs. We evaluate PF-RNNs on various real-world sequence prediction datasets across multiple domains for both regression and classification tasks. Regression tasks include stock index prediction (NASDAQ [32]), appliances energy prediction (AEP [4]), air quality prediction (AIR [8] and PM [25]). Classification tasks include activity recognition (UID [6], LPA [19], AREM [31] and GAS [14]) and text classification (R52 [5] and MR [26]). For regression tasks, a prediction is made after each time step, while for classification tasks prediction is made at the end of each sequence using the belief at the last time step.

We compare PF-LSTM and PF-GRU with the corresponding LSTM and GRU models. As PF-RNNs possess a similar structure with gated RNNs, we can directly use them as drop-in replacements. We use a similar network architecture for all models and datasets. The input is passed through two fully-connected layers and input to the recurrent cell. The output of the recurrent cell is processed by another two fully connected layers to predict an output value. We perform grid search over standard training parameters (learning rate, batch size, gradient clipping value, regularization weight) for each model and dataset.

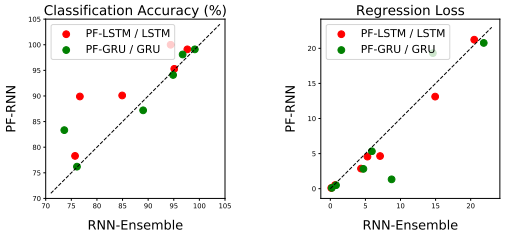

Figure 7: **Comparison with ensembles**. Scatter plot of classification accuracy (left) and regression loss (right) over all datasets. The x-axis is the performance of an RNN ensemble, the y-axis is the performance of a PF-RNN. The dashed line corresponds to equal performance. Note that PF-RNN is better for points above the dashed line for classification, and for points below the dashed line for regression. Red points compare PF-LSTM with LSTM ensemble, blue points compare PF-GRU with GRU ensemble.

We make two sets of comparisons. First, we fix the latent state size and number of particles, such that PF-RNNs and the corresponding RNNs have a similar number of parameters. Specifically, we use a latent state size of 50 and 20 particles for PF-LSTM and PF-GRU, and a latent state size of 80 for LSTM 86 and GRU. Second, we perform a search over these hyper-parameters independently for all models and datasets, and report the best-achieved result.

Results are shown in Table 1. We observe that PF-LSTM and PF-GRU are generally better than the LSTM and GRU, both when the number of parameters is comparable, as well as with the best hyper-parameters within the model class. PF-RNNs outperform the corresponding RNNs on 90% of the datasets across different domains, including classification and regression tasks.

Comparison with the State-of-the-Art. Achieving state-of-the-art (SOTA) performance is not the focus of this paper. Nevertheless, we include the SOTA results for reference, when available, in the last row of Table 1. We use the same training, validation and test sets as the SOTA methods.

Table 2: Ablation study.

	PF-LSTM									LSTM
	P1	P5	P10	P20	P30	NoResample	NoBNReLU	NoELBO	ELBOonly	BNReLU
Regression	0.81	0.66	0.59	0.55	**0.51**	0.76	0.87	0.68	0.73	0.74
Classification	85.88	87.55	91.25	92.12	**93.28**	89.92	88.55	90.70	88.60	86.28

PF-RNNs, despite having only simple vanilla network components, perform better than the SOTA on LPA, AREM, and GAS. For NASDAQ, R52 and MR, SOTAs [32, 39, 5] use complex network structure designed specifically for the task, thus they work better than PF-RNNs. Future work may investigate PF-RNNs with larger, more complex network components, which would provide a more fair comparison with the SOTA.

Comparison with Ensembles. PF-RNN can be considered as a method for improving an RNN, by maintaining multiple copies and using particle filter updates. Another common technique for improving the performance of a predictor using multiple copies is ensembling. We compare the performance of PF-RNN with bagging [2], an effective and commonly used ensembling method. Bagging trains each predictor in an ensemble by constructing a different training set of the same size through randomly sampling the original training set. We compare PF-RNNs using K particles, with ensembles of K RNNs. The results are summarized in Fig. 7. Detailed results are in the appendix.

Bagging reduces the variance of the predictors, but does not address belief tracking. Indeed, ensembles of RNNs generally improve over a single RNN, but the performance is substantially poorer than PF-RNNs for most datasets.

4.3 Ablation Study

We conduct a thorough ablation study on all datasets for PF-LSTMs to better understand their behavior. A summary of the results, averaged over all datasets, is shown in Table 2. Detailed results are in the appendix. To compute a meaningful average for regression tasks, we need values with the same order of magnitude. We rescale regression errors across the ablation methods into a range between 0 and 1 for each dataset.

We evaluate the influence of the following components: 1) number of particles (P1, P5, P10, P20, P30); 2) soft-resampling (NoResample); 3) replacing the hyperbolic tangent with ReLU activation and batch normalization (NoBNReLU); 4) combining prediction loss and ELBO for training (NoELBO and ELBOonly). We conduct the same, independent hyper-parameter search as in previous experiments.

We observe that: 1) Using more particles makes a better prediction in general. The general performance increases from PF-LSTM P1 to P30. 2) Soft-resampling improves the performance of PF-LSTM. 3) The combination of prediction loss and ELBO loss leads to better performance. 4) The ReLU activation with batch normalization helps to train PF-LSTM. Simply using ReLU with batch normalization does not make LSTM better than PF-LSTM.

5 Conclusion

PF-RNNs combine the strengths of RNNs and particle filtering by learning a latent particle belief representation and updating the belief with a particle filter. We apply the idea specifically to LSTM and GRU and construct PF-LSTM and PF-GRU, respectively. Our results show that PF-RNNs outperform the corresponding RNNs on various sequence prediction tasks, including text classification, activity recognition, stock price prediction, robot localization, etc.

Various aspects of the PF-RNN approach provide opportunities for further investigation. Currently, PF-RNNs make a prediction with the mean of the particle belief. An alternative is to aggregate particles, e.g., by estimating higher-order moments, estimating the entropy, attaching an additional RNN [15], or applying an attention mechanism [34].

PF-RNNs can serve as drop-in replacements for RNNs. While this work focuses on sequence prediction and classification, the idea can be applied equally well to sequence generation and sequence-to-sequence prediction, with application to, e.g., image captioning and machine translation.

References

[1] A. Blake and M. Isard. The condensation algorithm-conditional density propagation and applications to visual tracking. In *Advances in Neural Information Processing Systems*, pages 361–367, 1997.

[2] L. Breiman. Bagging predictors. *Machine learning*, 24(2):123–140, 1996.

[3] Y. Burda, R. Grosse, and R. Salakhutdinov. Importance weighted autoencoders. *arXiv preprint arXiv:1509.00519*, 2015.

[4] L. M. Candanedo, V. Feldheim, and D. Deramaix. Data driven prediction models of energy use of appliances in a low-energy house. *Energy and Buildings*, 140:81–97, 2017.

[5] A. Cardoso-Cachopo. Improving Methods for Single-label Text Categorization. PdD Thesis, Instituto Superior Tecnico, Universidade Tecnica de Lisboa, 2007.

[6] P. Casale, O. Pujol, and P. Radeva. Personalization and user verification in wearable systems using biometric walking patterns. *Personal and Ubiquitous Computing*, 16(5):563–580, 2012.

[7] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing*, pages 1724–1734, 2014.

[8] S. De Vito, E. Massera, M. Piga, L. Martinotto, and G. Di Francia. On field calibration of an electronic nose for benzene estimation in an urban pollution monitoring scenario. *Sensors and Actuators B: Chemical*, 129(2):750–757, 2008.

[9] P. Del Moral. Non-linear filtering: interacting particle resolution. *Markov processes and related fields*, 2(4):555–581, 1996.

[10] K. Gregor and F. Besse. Temporal difference variational auto-encoder. *arXiv preprint arXiv:1806.03107*, 2018.

[11] Z. D. Guo, M. G. Azar, B. Piot, B. A. Pires, T. Pohlen, and R. Munos. Neural predictive belief representations. *arXiv preprint arXiv:1811.06407*, 2018.

[12] T. Haarnoja, A. Ajay, S. Levine, and P. Abbeel. Backprop KF: Learning discriminative deterministic state estimators. In *Advances in Neural Information Processing Systems*, pages 4376–4384, 2016.

[13] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997.

[14] R. Huerta, T. Mosqueiro, J. Fonollosa, N. F. Rulkov, and I. Rodriguez-Lujan. Online decorrelation of humidity and temperature in chemical sensors for continuous monitoring. *Chemometrics and Intelligent Laboratory Systems*, 157:169–176, 2016.

[15] M. Igl, L. Zintgraf, T. A. Le, F. Wood, and S. Whiteson. Deep variational reinforcement learning for POMDPs. In *Proceedings of the International Conference on Machine Learning*, pages 2117–2126, 2018.

[16] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the International Conference on Machine Learning*, pages 448–456, 2015.

[17] R. Jonschkowski and O. Brock. End-to-end learnable histogram filters. In *NeurIPS Workshop on Deep Learning for Action and Interaction*, 2016.

[18] R. Jonschkowski, D. Rastogi, and O. Brock. Differentiable Particle Filters: End-to-End Learning with Algorithmic Priors. In *Proceedings of Robotics: Science and Systems (RSS)*, 2018.

[19] B. Kaluža, V. Mirchevska, E. Dovgan, M. Luštrek, and M. Gams. An agent-based approach to care in independent living. In *International Joint Conference on Ambient Intelligence*, pages 177–186. Springer, 2010.

[20] P. Karkus, D. Hsu, and W. S. Lee. QMDP-net: Deep learning for planning under partial observability. In *Advances in Neural Information Processing Systems*, pages 4694–4704, 2017.

[21] P. Karkus, D. Hsu, and W. S. Lee. Particle filter networks with application to visual localization. In *Proceedings of the Conference on Robot Learning*, pages 169–178, 2018.

[22] D. P. Kingma and M. Welling. Auto-encoding variational Bayes. In *Proceedings of the International Conference on Learning Representations*, 2014.

[23] T. A. Le, M. Igl, T. Rainforth, T. Jin, and F. Wood. Auto-encoding sequential Monte Carlo. In *Proceedings of the International Conference on Learning Representations*, 2018.

[24] M. Li, P. Xiao, and J. Zhang. Text classification based on ensemble extreme learning machine. *arXiv preprint arXiv:1805.06525*, 2018.

[25] X. Liang, T. Zou, B. Guo, S. Li, H. Zhang, S. Zhang, H. Huang, and S. X. Chen. Assessing Beijing's pm2.5 pollution: severity, weather impact, APEC and winter heating. *Proceedings of the Royal Society of London Series A*, 471(2182):20150257, 2015.

[26] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, 2011.

[27] C. J. Maddison, J. Lawson, G. Tucker, N. Heess, M. Norouzi, A. Mnih, A. Doucet, and Y. Teh. Filtering variational objectives. In *Advances in Neural Information Processing Systems*, pages 6573–6583, 2017.

[28] P. Moreno, J. Humplik, G. Papamakarios, B. Avila Pires, L. Buesing, N. Heess, and T. Weber. Neural belief states for partially observed domains. *NeurIPS Workshop on Reinforcement Learning under Partial Observability*, 2018.

[29] K. P. Murphy. *Dynamic Bayesian networks: representation, inference and learning*. PhD thesis, University of California, Berkeley, 2002.

[30] C. Naesseth, S. Linderman, R. Ranganath, and D. Blei. Variational sequential Monte Carlo. In *International Conference on Artificial Intelligence and Statistics*, pages 968–977, 2018.

[31] F. Palumbo, C. Gallicchio, R. Pucci, and A. Micheli. Human activity recognition using multisensor data fusion based on reservoir computing. *Journal of Ambient Intelligence and Smart Environments*, 8(2):87–107, 2016.

[32] Y. Qin, D. Song, H. Cheng, W. Cheng, G. Jiang, and G. W. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. In *Proceedings of the International Joint Conference on Artificial Intelligence*, pages 2627–2633, 2017.

[33] M. Ravanelli, P. Brakel, M. Omologo, and Y. Bengio. Light gated recurrent units for speech recognition. *IEEE Transactions on Emerging Topics in Computational Intelligence*, 2018.

[34] A. Santoro, R. Faulkner, D. Raposo, J. Rae, M. Chrzanowski, T. Weber, D. Wierstra, O. Vinyals, R. Pascanu, and T. Lillicrap. Relational recurrent neural networks. *Advances in Neural Information Processing Systems*, 2018.

[35] A. Somani, N. Ye, D. Hsu, and W. S. Lee. DESPOT: Online POMDP planning with regularization. In *Advances in Neural Information Processing Systems*, pages 1772–1780, 2013.

[36] S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial intelligence*, 128(1-2):99–141, 2001.

[37] P. J. Werbos. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 1990.

[38] W. Xiong, L. Wu, F. Alleva, J. Droppo, X. Huang, and A. Stolcke. The Microsoft 2017 conversational speech recognition system. In *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 5934–5938, 2018.

[39] P. Zhou, Z. Qi, S. Zheng, J. Xu, H. Bao, and B. Xu. Text classification improved by integrating bidirectional LSTM with two-dimensional max pooling. *arXiv preprint arXiv:1611.06639*, 2016.

A Derivation of Gradient Estimation

The PF-RNN predictor $\hat{y}_t$ is a randomized predictor, i.e., the value of $\hat{y}$ depends on random variables used for stochastic memory updates and soft-resampling. A reasonable objective function to minimize is $\mathrm{E}[L_{\mathrm{pred}}(\theta)] = \int_{h_p} p(h_p|\theta) \sum_{t \in \mathcal{O}} \ell(y_t, \hat{y}_t(h_p, \theta))$, where h_p denotes the set of random variables used in constructing the predictor. We have used the notation $\hat{y}_t(h_p, \theta)$ to make the dependence of $\hat{y}_t$ on h_p and the predictor parameter θ explicit; however, for notational convenience, we have left out the dependence on input x. After fixing h_p, $\hat{y}_t$ becomes a fixed function of θ and the input x which we use as our predictor; hence, we are optimizing the value of θ to generate good values of h_p as well as a good prediction once h_p has been generated.

For simplicity of notation, we consider only one term in the summation. The gradient is:

$$\begin{aligned}
\nabla_\theta \int_{h_p} p(h_p|\theta)\ell(y_t, \hat{y}_t(h_p, \theta)) &= \int_{h_p} \nabla_\theta (p(h_p|\theta)\ell(y_t, \hat{y}_t(h_p, \theta))) \\
&= \int_{h_p} (\nabla_\theta p(h_p|\theta))\ell(y_t, \hat{y}_t(h_p, \theta)) + \int_{h_p} p(h_p|\theta)\nabla_\theta \ell(y_t, \hat{y}_t(h_p, \theta)) \\
&= \int_{h_p} p(h_p|\theta)(\nabla_\theta \log p(h_p|\theta))\ell(y_t, \hat{y}_t(h_p, \theta)) \\
&\qquad + \int_{h} p(h_p|\theta)\nabla_\theta \ell(y_t, \hat{y}_t(h_p, \theta))
\end{aligned}$$

The product rule is used in the second line of the derivation; and identity $\nabla_\theta \log p(h_p|\theta) = (\nabla_\theta \log p(h_p|\theta))/p(h_p|\theta)$ is used in the last line. Sampling h_p and then computing $\nabla_\theta \log p(h_p|\theta)\ell(y_t, \hat{y}_t(h_p, \theta)) + \nabla_\theta \ell(y_t, \hat{y}_t(h_p, \theta))$ would give an unbiased estimator for the gradient of the objective function, as often required for stochastic gradient optimizers. However, the first term $\nabla_\theta \log p(h_p|\theta)\ell(y_t, \hat{y}_t(h_p, \theta))$ tends to have high variance [23]. As in other works, e.g. [23], we use only the second term $\nabla_\theta \ell(y_t, \hat{y}_t(h_p, \theta))$ to reduce the variance while introducing a small bias.

We now consider the variables in h_p for our problem. In the PF-RNN we obtain the value of the particle h_t^i at time t by sampling in the transition, conditioned on the parent h_{t-1}^i. We then do soft resampling which may replace the particle at position i with another particle which has a different parent.

For the transition, we use the *reparameterization trick* [22], which is able to reparameterize the problem so that the subset of randomization variables h_p that is involved in the transition becomes independent of θ. In our problem, we would like to learn the covariance of ξ_t^i. It has a zero mean Gaussian distribution with diagonal covariance, where the diagonal covariance is a learned function of h_{t-1}^i and x_t. As the covariance matrix is diagonal, we can consider each component separately. In the reparameterization trick, to draw a value v from a Normal distribution $\mathcal{N}(0, \sigma(h_{t-1}^i, x_t))$ with mean 0 and standard deviation $\sigma(h_{t-1}^i, x_t)$, we instead draw ε from $\mathcal{N}(0, 1)$ and output $v = \varepsilon\sigma(h_{t-1}^i, x_t)$. The value ε, which forms part of h_p can now be treated as part of the input to the RNN, making it independent of θ.

For soft resampling, the position of the replacement particle (and correspondingly its parent in the chain of ancestors) forms part of h_p. This depends on the parameter θ in $p(h_p|\theta)$ hence is affected by our approximation in the gradient estimate. We have not observed any problem attributable to the approximation in our experiments.

Finally, our predictor has the form $\hat{y}_t = f_{\mathrm{out}}(\bar{h}_t)$ where $\bar{h}_t = \sum_{i=1}^K w_t^i h_t^i$ and f_{out} if a learned function. The particles at time t are averaged, each particle at each time step (except the first step) has a parent that is constructed by the soft resampling operations, and the transitions have sampled inputs from the reparameterization operation. Our approximate gradient is computed by running the particle filter, forming the predictor, then computing the gradient of the loss of the predictor formed from the filter.

B Derivation of ELBO

As discussed in Section 3.3, we jointly optimize the prediction loss and an ELBO of $p(y_t|x_{1:t}, \theta)$. In standard maximum likelihood estimation, we aim to find the parameter θ that maximizes the log likelihood $\log p(y_t|x_{1:t}, \theta)$. This is often intractable. Instead, the evidence lower bound (ELBO) is often used, which is a variational lower bound of the log likelihood. We derive an ELBO using the technique from [3].

We first give a formal definition to the particle chain τ_t^i. We define τ_t^i in a recursive manner: $\tau_t^i = \tau_{t-1}^i \bigcup \{\epsilon_t^i, a_t^i\}$, where $\epsilon_t^i \sim \mathcal{N}(0, I)$ is the reparameterized random number used in the stochastic transition and a_t^i is the parent index chosen during soft-resampling. Given the input $x_{1:t}$, the particle chain determinizes the stochastic processes in PF-RNNs and produces a fixed particle h_t^i. In this case, sampling the particle chains $\tau_{1:t}^{1:K}$ conceptually gives K RNNs.

We first consider the simpler case of the particle chains being drawn independently. Assume that $p(y_t|x_{1:t}, \theta) = \int_{\tau_{1:t}} p(y_t, \tau_{1:t}|x_{1:t}, \theta)$. Then

$$\begin{aligned}
\log p(y_t|x_{1:t}, \theta) &= \log \int_{\tau_{1:t}} p(y_t, \tau_{1:t}|x_{1:t}, \theta) \\
&= \log \int_{\tau_{1:t}^1, \ldots, \tau_{1:t}^K} q(\tau_{1:t}^1|x_{1:t}) \ldots q(\tau_{1:t}^K|x_{1:t}) \frac{1}{K} \sum_{i=1}^K \frac{p(y_t, \tau_{1:t}^i|x_{1:t})}{q(\tau_{1:t}^i|x_{1:t})} \\
&\geq \int_{\tau_{1:t}^i, \ldots, \tau_{1:t}^K} q(\tau_{1:t}^1|x_{1:t}) \ldots q(\tau_{1:t}^K|x_{1:t}) \log \frac{1}{K} \sum_{i=1}^K \frac{p(y_t, \tau_{1:t}^i|x_{1:t}, \theta)}{q(\tau_{1:t}^i|x_{1:t})}
\end{aligned}$$

where we have used Jensen's inequality and assumed a variational distribution $q(\tau_{1:t}^1|x_{1:t}) \ldots q(\tau_{1:t}^K|x_{1:t})$ where $\tau_{1:t}^i$ are independently drawn from the same distribution.

We would like to use a particle filter to sample the particle chains for approximating the ELBO. The operations of our particle filter are as follows: at step t, we sample the transitions to generate new particles, we do resampling after the weights are updated by the observations, and we predict the target y_t. Due to the resampling operations, the particle chains in the particle filter are not independent. However, the particles generated at time t by the resampling operation are conditionally independent given the history of what has been generated before then; it is determined just by the ancestor indices a_t^i which are independently sampled given the particle chain distribution at that time by the resampling operation. Let Υ_t denote all the history before soft-resampling. Assuming $p(y_t|x_{1:t}, \theta) = \int_{\Upsilon_t, a_t} p(y_t, \Upsilon_t, a_t|x_{1:t}, \theta)$, $a_t \sim p$, where p is a multinomial distribution with $p(i) = w_t^i$,

$$\begin{aligned}
& \log \int_{\Upsilon_t, a_t} p(y_t, a_t, \Upsilon_t|x_{1:t}, \theta) \\
= & \log \int_{\Upsilon_t, a_t^1, \ldots, a_t^K} q(\Upsilon_t|x_{1:t}, \theta) q(a_t^1|\Upsilon_t, x_{1:t}, \theta) \ldots q(a_t^K|\Upsilon_t, x_{1:t}, \theta) \cdot \\
& \frac{1}{K} \sum_{i=1}^K \frac{p(y_t, a_t^i, \Upsilon_t|x_{1:t}, \theta)}{q(\Upsilon_t|x_{1:t}, \theta) q(a_t^i|\Upsilon_t, x_{1:t}, \theta)} \\
\geq & \int_{\Upsilon_t, a_t^1, \ldots, a_t^K} q(\Upsilon_t|x_{1:t}, \theta) q(a_t^1|\Upsilon_t, x_{1:t}, \theta) \ldots q(a_t^K|\Upsilon_t, x_{1:t}, \theta) \cdot \\
& \log \frac{1}{K} \sum_{i=1}^K \frac{p(y_t, a_t^i, \Upsilon_t|x_{1:t}, \theta)}{q(\Upsilon_t|x_{1:t}, \theta) q(a_t^i|\Upsilon_t, x_{1:t}, \theta)}
\end{aligned}$$

We now use the variational distributions $q(a_t^i|\Upsilon_t, x_{1:t}, \theta) = p(a_t^i|\Upsilon_t, x_{1:t}, \theta)$ and $q(\Upsilon_t|x_{1:t}, \theta) = p(\Upsilon_t|x_{1:t}, \theta)$. This allows the simplification $\frac{p(y_t, a_t^i, \Upsilon_t|x_{1:t}, \theta)}{p(a_t^i|\Upsilon_t, x_{1:t}, \theta) p(\Upsilon_t|x_{1:t}, \theta)} = p(y_t|a_t^i, \Upsilon_t, x_{1:t}, \theta)$. We opt for simplicity in the ELBO; the use of other variational distributions may allow the weights w_t^i to be optimized, as well as to incorporate information about y_t in the variational distribution, but at the expense of optimizing a more complex objective. We rely on L_{pred} to optimize w_t^i, which is fully task-oriented.

After negation (to be consistent with minimizing loss), we get our objective function

$$-\int_{\Upsilon_t, a_t^i, \ldots, a_t^K} p(\Upsilon_t|x_{1:t},\theta)p(a_t^1|\Upsilon_t,x_{1:t},\theta)\ldots p(a_t^K|\Upsilon_t,x_{1:t},\theta)\log\frac{1}{K}\sum_{i=1}^{K} p(y_t|a_t^i,\Upsilon_t,x_{1:t},\theta).$$

We then sample $\Upsilon_t, a_t^1, \ldots, a_t^K$ from $p(\Upsilon_t|x_{1:t},\theta)p(a_t^1|x_{1:t},\theta)\ldots p(a_t^K|x_{1:t},\theta)$ to get $\log\frac{1}{K}\sum_{i=1}^{K} p(y_t|a_t^i,\Upsilon_t,x_{1:t},\theta)$ and differentiate to get an estimator for the gradient.

Computing the derivative:

$$\begin{aligned}\nabla_\theta \log\frac{1}{K}\sum_{i=1}^{K} p(y_t|a_t^i,\Upsilon_t,x_{1:t},\theta) &= \frac{1}{\frac{1}{K}\sum_{i=1}^{K} p(y_t|a_t^i,\Upsilon_t,x_{1:t},\theta)}\nabla_\theta(\frac{1}{K}\sum_{i=1}^{K} p(y_t|a_t^i,\Upsilon_t,x_{1:t},\theta)) \\ &= \sum_{i=1}^{K}\frac{p(y_t|a_t^i,\Upsilon_t,x_{1:t},\theta)}{\sum_{i=1}^{K} p(y_t|a_t^i,\Upsilon_t,x_{1:t},\theta)}\nabla_\theta \log p(y_t|a_t^i,\Upsilon_t,x_{1:t},\theta).\end{aligned}$$

As in Appendix A, the parameter θ also affects the sampling distribution $p(\Upsilon_t|x_{1:t},\theta)$ of the ELBO. To get an unbiased estimate of the gradient, we will need to compute $(\nabla_\theta p(\Upsilon_t|x_{1:t},\theta)p(a_t^1|x_{1:t},\theta)\ldots p(a_t^K|x_{1:t},\theta))\log\frac{1}{K}\sum_{i=1}^{K} p(y_t|a_t^i,\Upsilon_t x_{1:t},\theta) + \nabla_\theta \log\frac{1}{K}\sum_{i=1}^{K} p(y_t|a_t^i,\Upsilon_t,x_{1:t},\theta)$ after sampling $\Upsilon_t, a_t^1, \ldots, a_t^K$. Again, we only compute $\nabla_\theta \log\frac{1}{K}\sum_{i=1}^{K} p(y_t|a_t^i,\Upsilon_t,x_{1:t},\theta)$, obtaining a lower variance but biased estimator of the gradient.

Finally we have $p(y_t|a_t^i,\Upsilon_t,x_{1:t},\theta) = p(y_t|\tau_{1:t}^i,x_{1:t},\theta)$ according to our definition, since we are able to do the prediction with only the knowledge of the single RNN. As mentioned in the Section 3.3, we compute $p(y_t|\tau_{1:t}^i,x_{1:t},\theta)$ by applying f_{out} on each particle h_t^i, assuming cross-entropy for classification tasks and unit variance Gaussian model for regression, gradients are carried through the chain of particle states through time by BPTT.

C PF-GRU Network Architecture

The standard GRU maintains the latent state $\hbar_t$, and updates it with two gates: reset gate r_t and update gate z_t. The memory is updated according to the following equations:

$$\hbar_t = (1-z_t)\circ\tanh(n_t) + z_t\circ\hbar_{t-1}, \qquad n_t = W_n[r_t\circ\hbar_{t-1},x_t] + b_n \tag{9}$$

where W_n and b_n are the corresponding weights and biases.

The state of the PF-GRU is a set of weighted particles $\{\hbar_t^i, w_t^i\}_{i=1}^K$. Similarly to PF-LSTM, we perform a stochastic cell update, where the update to the cell, n_t^i, is sampled from a Gaussian distribution:

$$n_t^i = W_n[r_t^i\circ\hbar_{t-1}^i,x_t] + b_n + \epsilon_t^i, \quad \epsilon_t^i\sim\mathcal{N}(0,\Sigma_t^i), \quad \Sigma_t^i = W_\Sigma[\hbar_{t-1}^i,x_t] + b_\Sigma \tag{10}$$

Besides, to allow training with longer truncated BPTT, similarly to PF-LSTM, the hyperbolic tangent function is replaced by ReLU with batch normalization:

$$\hbar_t^i = (1-z_t^i)\circ\text{ReLU}(\text{BN}(n_t^i)) + z_t\circ\hbar_{t-1}^i \tag{11}$$

D Experimental Setup Details

We implement all models in PyTorch and train using NVidia GTX1080Ti GPUs.

For all models, we perform a grid search over standard training parameters (learning rate, batch size, gradient clipping value, regularization weight). Specifically, we perform a standard grid search over the learning rate $\{1^{-4}, 3^{-4}, 5^{-4}\}$ with optimizer Adam and RMSProp, batch size $\{16, 32, 64, 128\}$, gradient clipping value $\{3, 5\}$ and L2 regularization weight of $\{0.001, 0.0001\}$. When model hyperparameter search is performed, for PF-LSTM/PF-GRU it is a simple grid search over $\{64, 128, 256\}$ latent state sizes and $\{20, 30\}$ particles. For LSTM/GRU we do a more careful search, and increase the latent state size, $\{80/86, 64, 128, 256, 512, 1024, \ldots\}$, until the performance stops improving.

E Additional Results

E.1 Comparison with RNN Ensemble Models

The detailed results for the comparison with RNN ensemble models is given in Table 3 and Table 4. LSTM-E/GRU-E denote the ensemble models of K LSTMs/GRUs. We compare with PF-LSTM/PF-GRU with K particles. For the localization experiments, i.e., Maze 10, Maze 18 and Maze 27 rows, we use $K = 30$ ensemble models/particles. For the rest, we use $K = 20$.

Table 3: PF-RNNs V.S. Bagging. Prediction (%)

	LSTM-E	PF-LSTM	GRU-E	PF-GRU
LPA	94.4	**100.0**	96.7	**98.1**
AREM	97.7	**99.1**	99.1	**99.1**
GAS	76.7	**89.9**	73.6	**83.3**
UID	95.1	**95.3**	**94.9**	94.1
R52	84.9	**90.1**	**89.0**	87.2
MR	75.7	**78.3**	76.1	**76.2**

Table 4: PF-RNNs V.S. Bagging. Regression

	LSTM-E	PF-LSTM	GRU-E	PF-GRU
AEP	5.29	**4.57**	5.92	**5.33**
NASDAQ	7.09	**4.65**	8.73	**1.33**
AIR	14.94	**13.12**	**14.61**	19.32
PM	**20.51**	21.23	21.86	**20.77**
Maze 10	0.13	**0.10**	0.21	**0.11**
Maze 18	0.67	**0.51**	0.81	**0.49**
Maze 27	4.37	**2.86**	4.71	**2.82**

E.2 Ablation Study

We present the detailed results for our ablation study over all datasets in Table 5 and Table 6.

Table 5: Ablation Study for Regression Datasets

	Maze 10	Maze 18	Maze 27	AEP	AIR	PM	NASDAQ
PF-LSTM-P1	0.41	1.1	3.84	6.24	15.41	22.12	13.21
PF-LSTM-P5	0.17	0.65	3.81	5.73	13.89	21.98	10.97
PF-LSTM-P10	0.12	0.6	3.11	4.88	14.23	21.33	9.25
PF-LSTM-P20	0.12	0.55	3.03	4.57	**13.12**	21.23	4.64
PF-LSTM-P30	**0.103**	**0.51**	**2.862**	**3.24**	14.44	**20.05**	**4.55**
PF-LSTM-NoResample	0.14	1.42	4.03	5.27	18.47	20.24	21.23
PF-LSTM-NoBNReLU	0.19	1.66	4.01	5.03	16.73	21.01	51.32
PF-LSTM-NoELBO	0.18	1.23	3.21	4.88	16.89	21.79	6.59
PF-LSTM-ELBOonly	0.34	1.12	3.01	4.94	17.15	21.11	9.87
LSTM-BNReLU	0.21	1.05	4.15	5.35	16.29	21.38	15.72

Table 6: Ablation Study for Classification Datasets

	UID	R52	LPA	AREM	GAS	MR
PF-LSTM-P1	89.7	86.4	97.1	98.1	71.3	72.7
PF-LSTM-P5	94.9	88.1	96.9	98.4	71.8	75.2
PF-LSTM-P10	94.7	88.7	97.7	98.4	89.2	78.8
PF-LSTM-P20	95.3	**90.1**	**100**	**99.1**	89.9	78.3
PF-LSTM-P30	**98.4**	88.5	**100**	98.1	**94.1**	**80.6**
PF-LSTM-NoResampe	95	85.2	95.3	99	88.7	76.3
PF-LSTM-NoBNReLU	95.7	84.8	96.2	98.6	78.9	77.1
PF-LSTM-NoELBO	94.2	88.1	97.7	97.2	90.8	76.2
PF-LSTM-ELBOonly	91.9	87.8	97.3	97	81.4	76.2
LSTM-BNReLU	91.2	82.3	97.7	98.9	75.7	71.9

E.3 Visualization of Additional Examples for Localization