Use Particle Filter to Train a Neural Network

Chen-Yi Liu

University of Manitoba

May 10, 2020

Introduction

The most widely used method to train a feed forward or a recurrent artificial neural network is with the use of back propagation. This approach has shown great success as evident in the now myriad of deep neural networks in the world being trained this way. However, back propagation initially faced a few challenges after its introduction in the 1960's [1].

Part of the reasons that the back propagation algorithm stumbled in its early days was that the neuron activation functions used at that time were mostly the sigmoid function and the hyperbolic tangent function. The use of sigmoid and hyperbolic tangent function caused the vanishing gradient problem when used in recurrent network or in multi-layer feed forward network. Nowadays, the vanishing gradient problem has been largely circumvented by the use of Lon-Short-Term-Memory (LSTM) in recurrent network and the use of Rectified Linear Unit (ReLU) activation function in multi-layer feed forward network.

Before these work around methods were found, researchers in the past have tried using other algorithms to train a neural network [1]. One of these attempts used recursive Bayesian estimator to estimate the parameters in a neural network. The two prominent Bayesian estimators that have been used to train neural networks were Extended Kalman Filter and Particle Filter.

Built on the success of Kalman Filter, the Extended Kalman Filter was developed to estimate parameters involved in non-linear functions using locally linear approximations. People have demonstrated the ability of Extended Kalman Filter to train a recurrent neural network [1]. On the other hand, Particle filter is built from the start to handle non-linear functions. This characteristic makes Particle filter a good candidate algorithm to estimate the parameters of a neural network. In fact, people have shown that it is possible to train a small neural network using particle filter [2].

In our project, we decided to use particle filter to train a neural network. If the method turns out successful, it will open up the possibility of using any activation function in the neural network. People will no longer be restricted to use partially linear activation functions such as ReLU. People will also be able to build recurrent neural networks using neurons that may be much simpler than LSTM.

Methods

Particle filter (without resample)

We follow the approach of [2] to use particle filter to train a neural network. As a standard to compare with, we also trained the neural network using backpropagation and stochastic gradient descent (SGD).

The algorithm of particle filter neural network training is as follows. The estimated parameters of a neural network at training step, k, is represented as x_k. Because the training data distribution is assumed to be stationary, the transition model of the parameters is

$$x_{k+1} = x_k + d_k$$

, where d_k is a Gaussian noise with zero mean.

At each training step k, the performance of a particle i on the training data is denoted by the importance ratio of the particle, $I_k(i)$. The importance ratio is calculated as the softmax of $L_{k,i}$, the loss value of particle i at step k,

$$I_k(i) = \frac{e^{-L_{k,i}}}{\sum_{j=0}^{N} e^{-L_{k,j}}}$$. N is the number of particles. The smaller the loss value, the higher the importance ratio value. The importance ratios sum to 1.

The importance ratio at each time step is an independent statistical event from those of other steps. Therefore, the joint probability of the optimal parameters is the product of importance ratios from all steps. After normalization, we get the below formula for the cumulative importance ratio of particle i.

$$I(i) = \frac{\prod_{l=0}^{k} I_l(i)}{\sum_{j=0}^{N} \prod_{l=0}^{k} I_l(j)}$$

At the end of training, the particle with the highest importance ratio is picked as the trained network. This corresponds to the Maximum A Posteriori estimate of the parameters of the network.

Particle filter (with resample)

The importance ratios of all particles and their locations in parameter space together form a probability distribution function. This function is the probability distribution of the optimal neural network parameters. It is possible to sample from this distribution and make a new generation of particles.

The resample is not done at every training step. Resample is only performed when the effective number of particles falls below a threshold. The effective number of particles is calculated as

$N_{effective} = \frac{1}{\sum_{i=1}^{N} (I(i))^2}$, and consequently $1 \leqslant N_{effective} \leqslant N$. The threshold needs to be set to balance between having adequate coverage of the parameter space and having enough particles in the region of highest probability. The threshold in this project is set to 60% of the total number of particles. As a result, a new generation of particles is generated, and the old ones discarded, whenever the effective number of particles falls below 60% of the total number of particles.

Backpropagation

The backpropagation algorithm was used here as a standard to compare to. The standard stochastic gradient descent with a mini batch size of 1 was used. No augmentation such as momentum or Adam was used. Two learning rates 0.1 and 0.01 were evaluated.

Code

The particle filter algorithm was written in Python 3 using the NumPy library. The neural network was built with the PyTorch library. Backpropagation was done using PyTorch's auto differentiation function. The code is hosted on https://github.com/chenyi5759/particle_filter

Main challenges

One of the challenges to use particle filter to train a neural network is handling particles in an efficient way. The PyTorch library does not have a function to train multiple copies of a neural network simultaneously. One needs to use a for loop to call PyTorch functions to either do a forward pass or a backward pass through each copy of the neural network. The overhead involved in calling PyTorch functions repeatedly prevented us from using large numbers of particles, >100.

Therefore, to use particle filter without resample, we implemented the neural network using plain Python functions and NumPy. This approach involves less overhead and allows us to train groups of up to 500,000 particles.

Particle filter with resample does not require a large amount of particles, so it is less sensitive to this problem.

Results and discussion

We implemented two particle filter algorithms (with and without resample) to train a feed forwad fully connected neural network. And, we compared training results to those from backpropagation. The network consists of a single neuron with two inputs. The activation function of the neuron was the sigmoid function, and the network was used as a binary classifier with binary cross entropy loss. The training data were the Kids Height dataset.

Experiment 1

The first experiment was done to make sure that the particle filter algorithm is capable of training a neural network. For this purpose, we compare the values of the two weights and bias of the neural network trained with particle filter and backpropagation. We plotted the results as a separating line among the data points from the training dataset. As can be seen in Figure 1 and Figure 2, the results were very similar. Both training methods produce network parameters that can clearly separate the data points into the correct categories.

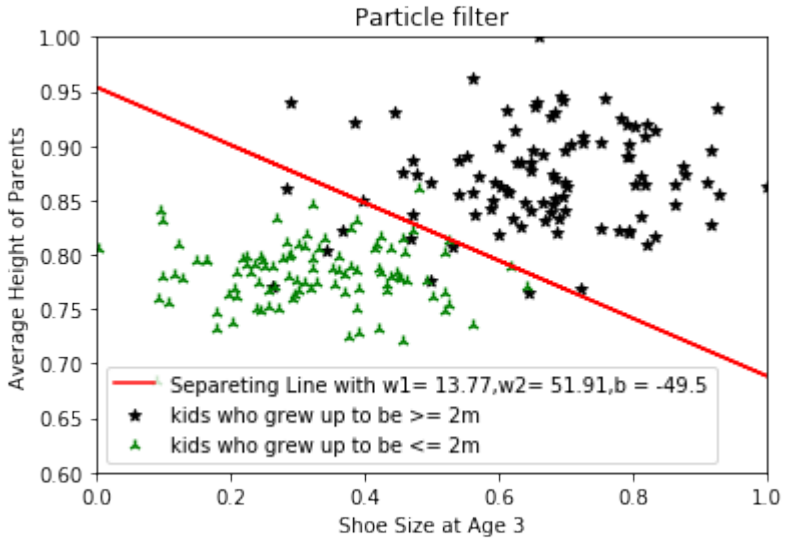

Figure 1: Binary classifier trained with Particle Filter (without resample).

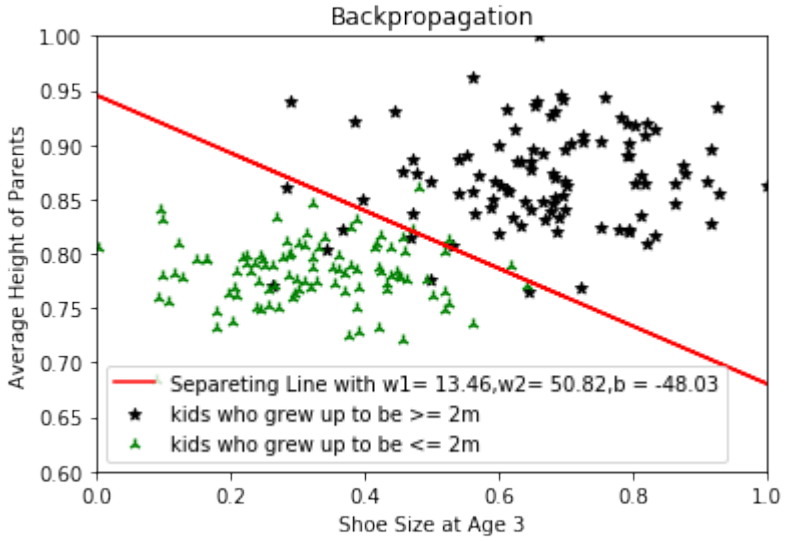

Figure 2: Binary classifier trained with backpropagation.

Experiment 2

After the first experiment, we knew that it was possible to use particle filter to train a neural network. Now the question becomes whether particle filter is better or worse than backpropagation at training. Knowledge of this can help us decide which method to use for future neural network training. One of the metric to compare training methods is the amount of time it takes to train the network. In other words, we compared the number of training steps required for the algorithm to arrive at the correct set of parameters. In particular, we looked at the number of training steps required to get the best performing particle to reach 90% classification accuracy.

For the particle filter without resample, we see in Figure 3 that the more particles we use, the faster we reach the optimal values of the parameters. This was not too surprising. The particle filter without resample was basically guessing the parameters. So, the more guesses the algorithm can make at one step, the sooner for it to make the correct guess.

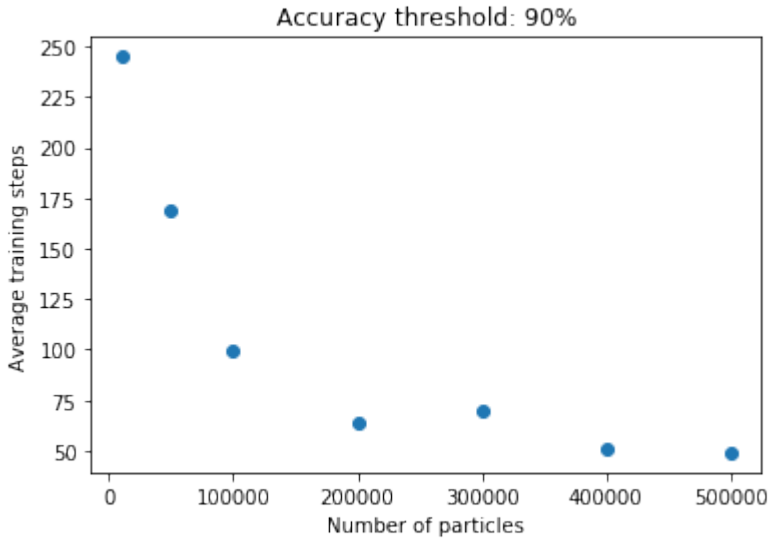

Figure 3: Number of training steps decreases with the number of particles. The first two data points are 10,000 and 50,000 particles. Each data point is an average over 5 trials.

Next, we looked at the particle filter with resample, Figure 4. First of all, it requires much fewer particles compared to the version of particle filter without resample. When compared to backpropagation, particle filter is slightly slower to both learning rate 0.1 and 0.01.

We train multiple neural networks at the same time with backpropagation. We observe that the more neural networks we train simultaneously, the faster we reach the optimal parameters. This is probably because some of the parameters, when they were initialized, they were already close to the optimal values. So, those particles will need very few training steps to reach the minimum.

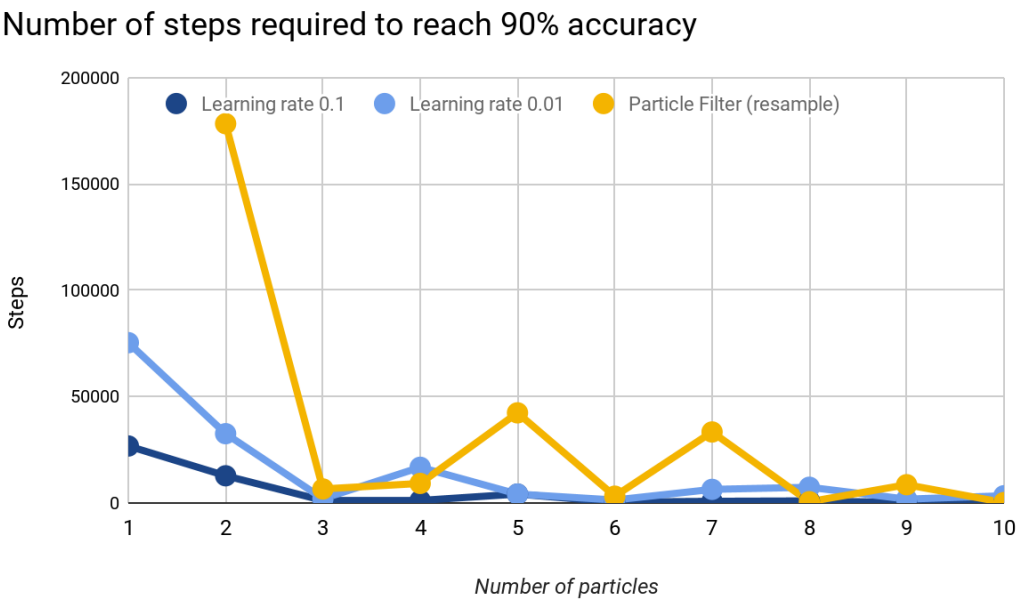

Figure 4: When there is a sufficient number of particles, it is quicker to reach the optimal values of the parameters. Learning rate 0.1 and 0.01 are done with backpropagation.

Also, as expected, the smaller learning rate does increase the number of steps required. The curve has bumps and is not a smooth curve showing the perfect inverse relationship. This is because the weights were generated randomly, and so there is some level of randomness in our value shown here.

One thing to note is that, if there is only one particle, resampling does not work. If there is only one particle, we cannot do comparison, and so the particle just does random walk in the parameter space. Thus, a minimum of 2 particles is needed to find the gradient. Resampling is like stochastic gradient descent without backpropagation. The gradient information is collected from the importance ratios of all the particles.

Resampling also resembles evolutionary algorithm, where the importance ratio can be seen as the fitness score or the particle.

Findings of this project

1. Particle filter can train a neural network.
2. Particle filter (without resample) trains by guessing the parameters, and is inefficient.
3. Particle filter (with resample) is slightly less efficient than backpropagation.
4. Resampling is like performing gradient descent.

Conclusion and future work

We demonstrated the ability of particle filter (with / without resample) to train a simple neural network. For the network tested in this project, particle filter (with resample) is slower at training than backpropagation.

A possible future work is to implement a hybrid algorithm that combines backpropagation with resampling as shown in [2]. The authours showed that the hybrid algorithm out performs either backpropagation or resampling alone. Another possible future work is to use the particle filter with resample to train more complicated neural networks such as deep convolutional neural networks or recurrent neural networks. The training may require more particles and more computing resources.

Acknowledgment

This report was the result of a class project in ECE7650 – Deep Learning with CNNs course taught at the University of Manitoba in the winter semester of 2020. The project was done in collaboration with a fellow student Junyao Pu. Both Junyao and I independently performed experiment 1 to confirm the result. Junyao did the particle filter (without resample) part of experiment 2, and I did the particle filter (with resample) and the backpropagation part of experiment 2.

References

1. Haykin S. Kalman filtering and neural networks. John Wiley & Sons; 2004 Mar 24.
2. Freitas JD, Niranjan M, Gee AH, Doucet A. Sequential Monte Carlo methods to train neural network models. Neural computation. 2000 Apr 1;12(4):955-93.