Movement Prediction Using Bayesian Learning for Neural Networks

Sherif Akoush
Department of Computer Science,
The American University in Cairo,
P.O.Box 2511, Cairo, Egypt
Email: sameh@aucegypt.edu

Ahmed Sameh
Department of Computer Science,
The American University in Cairo,
P.O.Box 2511, Cairo, Egypt
Email: sameh@aucegypt.edu

Abstract

*A technique for reducing the wireless cost of tracking mobile users with uncertain parameters is developed in this paper. Such uncertainty arises naturally in wireless networks, since an efficient user tracking is based on a prediction of its future call and mobility parameters. The conventional approach based on dynamic tracking is not reliable in the sense that inaccurate prediction of the user mobility parameters may significantly reduce the tracking efficiency. Unfortunately, such uncertainty is unavoidable for mobile users, especially for a burst mobility patterns. In this paper, we present a novel hybrid Bayesian neural network model for predicting locations on Cellular Networks (can also be extended to other wireless networks such as WI-FI and WiMAX). We investigate different parallel implementation techniques on mobile devices of the proposed approach and compare it to many standard neural network techniques such as: Back-propagation, Elman, Resilient, Levenberg-Marqudat, and One-Step Secant models. Bayesian learning for Neural Networks predicts location better than standard neural network techniques since it uses well founded probability model to represent uncertainty about the relationships being learned. The result of Bayesian training is a posterior distribution over network weights. We use Markov chain Monte Carlo methods (**MCMC**) to sample N values from the posterior weights distribution.*

1 Introduction

If a network can predict where the user is, then considerable bandwidth can be saved and resources can be optimized in mobility management. Prediction is regarded as one of the direct application of artificial intelligent systems. Usually we have large amount of sensor data that need to be interpreted in order to extract knowledge from this information. Such knowledge can be extremely useful to optimize resources and provide intelligent services. We usually try to learn (extract) patterns from the available data. Patterns can be associative such as attributes that occur together, classification such as indication of a given category or temporal such as sequences that happen frequently [1]. Prediction attempts to form patterns that permit it to predict the next event(s) given the available input data. In this paper, focus is on predicting user movements in wireless networks. In other words, we would like to predict what are the next locations the users would probably be given their past movements. Predicting movements in such domain is essential as it will enable the network to effectively allocate resources, better location update procedures and easier location search techniques [2]. This work can also be extended to even predict what services the user is expected to use are. Consequently, better quality of services is provided and the network is able to allocate efficiently the needed resources not only at the right place but also at the right time. We want to reduce the number of explicit locations updates/paging if we can successfully predict where the user is. In other words, if the system can predict accurately where the user is roaming, the user does not have to update the network about his location (location update). Moreover, the core network does not have to search where the user is (location paging). Therefore we are reducing the number of overall updates/paging messages which take considerable network bandwidth. If services that the user is expected to use are predicted, the network can plan efficiently resources allocations. This will ensure of course quality of services provided by the network. It is interesting to experiment whether services that the user is using are related to his movement patterns. There are several techniques that can be used in movement predictions such as Artificial Neural Networks, Bayesian Belief Networks, Hidden Markov Chains, Dynamic Belief Networks…etc [1]. Each technique has its advantages and disadvantages. Our work uses a hybrid technique – Bayesian Neural

Second International Conference on Systems and Networks Communications (ICSNC 2007)
0-7695-2938-0/07 $25.00

Networks that makes use of the pros of Bayesian inference in Artificial Neural Networks [4]. Section 2 describes Bayesian learning for neural networks, section 3 describes the experimentation methodology, and section 4 draws the collusion.

2 Bayesian Learning for Neural Networks

MLP Networks are widely used flexible models suitable in complex nonlinear problems [3]. The main questions in MLP models are the estimation of the model parameters and controlling the model complexity. The optimal number of degrees of freedom in the model depends on the number of training samples, amount of noise in the samples and the complexity of the underlying function being estimated. With standard neural networks techniques, to determine the correct model complexity and to set up a network with the desired complexity are often computationally very expensive. Another problem of standard neural network methods is the lack of tools for analyzing the results (confidence intervals for the results, like 10 % and 90 %). Recently Bayesian methods have become a viable alternative to the older error minimization based approaches. Researchers have suggested the importance of incorporating human knowledge in neural networks model to improve their performance. This knowledge is modeled through prior distribution over Neural Networks parameters. Bayesian methods use probability to quantify uncertainty in inferences and the result of Bayesian learning is a probability distribution expressing our beliefs regarding how likely the different predictions are. Predictions are made by integrating all models over this posterior distribution. The Bayesian learning uses probability to represent uncertainty about the relationship being learned. Before data is seen, prior opinions about what the true relationship might be can be expressed in a probability distribution over the network weights that define this relationship. After the training data is presented, the revised opinions are captured by a posterior distribution over network weights. Network weights that seemed plausible before, but which do not match the data very well, will now be seen as being much less likely, while the probability for values of the weights that do fit the data well will be increased [1]. In Bayesian data analysis all uncertain quantities are modeled as probability distributions, and inference is performed by constructing the posterior conditional probabilities for the unobserved variables of interest, given the observed data sample and prior assumptions.

$$P(\text{parameters}\,|\,\text{data}) = \frac{P(\text{parameters})\,P(\text{data}\,|\,\text{parameters})}{P(\text{data})}$$

P(paramerters|data) is the posterior probability of parameters, P(parameters) is the prior probability, P(data|parameters) is the likelihood and P(data) is a normalization constant. We want to make use of probability theories in Neural Networks as it will naturally overcome the possible problems in traditional learning. One of the major benefits of Bayesian MLP is that the resulting prediction is an average prediction of possible MLP solutions weighted by their probability. In other words, Bayesian MLP returns theoretically all possible solutions and integrates them out. Traditional MLP can be regarded as specific solution from the set returned by the Bayesian one. If the inputs of the network are set to the values for some new case, the posterior distribution over network weights will give rise to a distribution over the outputs of the network, which is known as the predictive distribution for this new case. If a single-valued prediction is needed, one might use the mean of the predictive distribution, but the full predictive distribution also tells how uncertain this prediction is. In Bayesian MLP:

$$P(\text{new data}\,|\,\text{data}) = \int_{\text{parameters}} P(\text{new data}\,|\,\text{parameters})\,P(\text{parameters}\,|\,\text{data})$$

P(new data|data) is the solution, P(new data|parameters) is the traditional MLP function and P(parameters|data) is the probability of this specific MLP function. The solution is thus integration over all possible MLP solutions (weights, bias…etc). Implementing the exact model is one of the biggest problems with Bayesian methods. Dealing with a complex distribution over weights is not as simple as finding a single "best" value for the weights. We have to drawn random samples and average is:

$$\hat{y}^{\text{new}} \approx \frac{1}{N}\sum_{t=1}^{N} f(x^{\text{new}}|\theta^{(t)}, M).$$

Where N different samples for possible MLP solutions (weights, bias…etc) θ [4]. Y new is calculated by taking the mean of the N different MLP outputs. M represent all parameters defining the model such as the number of perceptrons in the hidden layer and the choice of the activation function. In full Bayesian implementation we have to take into consideration different models M (varying the number of hidden perceptrons). Consequently, the final output is also the average of all models M outputs.

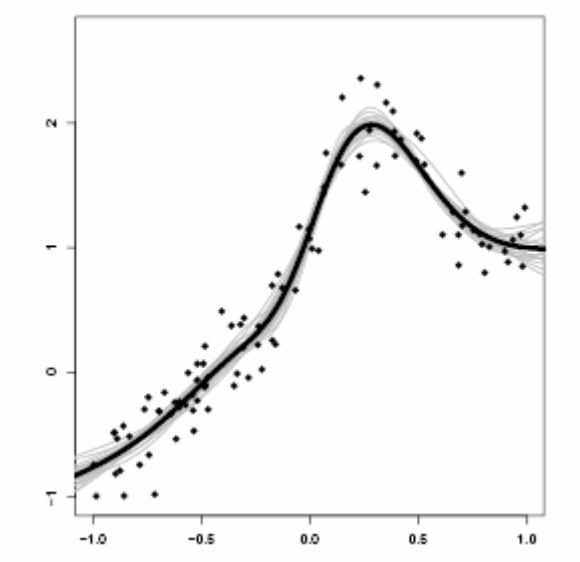

Figure 1: Bayesian MLP Solution

Figure 1 describes the solution of Bayesian MLP solution in a regression problem. The dots are the data points. The gray thin lines are N different solutions and the thick solid line is the average solution. It is easy to spot that the average solution is smoother that some individual solutions. In case of MLP the posterior distribution is typically very complex. The integrations required by Bayesian approach can be approximated using Markov Chain Monte Carlo (MCMC) methods [6]. An integral $\mu = \int g(x)p(x)\,dx$ can be approximated by MCMC, using a sample of values x(t) drawn from the distribution:

$$\hat{\mu}_n \approx \frac{1}{N}\sum_{t=1}^{N} g(x_{(t)})$$

Samples are generated using a Markov chain that has the desired posterior distribution as its stationary distribution. Monte Carlo methods for Bayesian neural networks have been developed by Neal [2]. The posterior distribution is represented by a sample of perhaps a few dozen sets of network weights. The sample is obtained by simulating a Markov chain whose equilibrium distribution is the posterior distribution for weights. The key idea in MCMC methods is to obtain a sample from the posterior and then base inference on that sample, for example, replacing posterior expectations with sample means over the simulated posterior sample. The main difficulty is MCMC methods is in generating a sample from the posterior p(θ|D). The rational is to consider a Markov chain {θn} with state θ and having p(θ|D) as stationary distribution. The strategy is to start with arbitrary values θ, let the Markov chain run until it has practically reached convergence, say after T iterations, and use the next k observed values of the chain as an approximate posterior sample A = { θ1, θ2..., θk}. In other words, the state of the chain after a large number of steps is then used as a sample from the desired distribution. The quality of the sample will improve as a function of the number of steps. Usually it is not hard to construct a Markov Chain with the desired properties. The more difficult problem is to determine how many steps are needed to converge to the stationary distribution within an acceptable error. There are several algorithms used to implement MCMC methods such as Metropolis-Hastings sampling, Hybrid Monte Carlo sampling, Gibbs sampling and Reversible jump Markov chain Monte Carlo sampling [6]. The method is exact in the limit as the size of the sample and the length of time for which the Markov chain is run increase, but convergence can sometimes be slow in practice.

3 Experimental Methodology

The proposed approach usually requires more expert work than the standard approach, either to devise reasonable assumptions for the distributions, or to include different options in the models and integrate them, but once that is done, the results are consistently better than with other approaches. There are several packages available to implement Bayesian Learning for Neural Networks. We choose MCMCStuff from Helsinki University (Finland) [3] because it implements all Bayesian methods for MLP in the Matlab environment. MCMCstuff toolbox is a collection of Matlab functions for Bayesian inference with Markov chain Monte Carlo (MCMC) methods. MCMCstuff toolbox is a collection of Matlab functions for Bayesian inference with Markov chain Monte Carlo (MCMC) methods [6]. The MCMC methods for MLP package has been written using Matlab and C programming languages and works with Matlab versions 6.* and 7.* as a toolbox. The code in this toolbox has been streamlined and optimized for faster computation. Some of the most computationally critical parts have been coded in C. This toolbox provides different sampling methods to implement MCMC such as Metropolis-Hastings sampling, Hybrid Monte Carlo sampling, Gibbs sampling and Reversible jump Markov chain Monte Carlo sampling. We have tested this toolbox for MLP network in regression problem with Gaussian noise. We have also tested the same regression problem with conventional Artificial Neural Network with backpropagation learning. Results show that Bayesian learning for Neural Network generalizes better.

We have downloaded dataset collected by the Reality Mining Project at MIT [5]. The Reality Mining project represents the largest mobile phone experiment ever

attempted in academia. The project is collecting an unprecedented amount of data on human behavior and group interactions that has been anonymized and made available to the general academic community. This dataset contain over 500,000 hours (~60 years) of continuous data on daily human behavior. The dataset has been used by researchers in a wide range of fields (including epidemiology, sociology, physics, artificial intelligence, and organizational behavior). The dataset is collected using one hundred Nokia 6600 smart phones using a version of the Context application from the University of Helsinki. Seventy-five users are either students or faculty in the MIT Media Laboratory, while the remaining twenty-five are incoming students at the MIT Sloan business school adjacent to the laboratory. Of the seventy-five users at the lab, twenty are incoming masters students and five are incoming MIT freshman. The information collected includes call logs, Bluetooth devices in proximity, cell tower IDs, application usage, and phone status (such as charging and idle), which comes primarily from the Context application. The study generated data collected by one hundred human subjects over the course of nine months and represent approximately 500,000 hours of data on users' location, communication and device usage behavior. The portion of that dataset that is most important to our experiments is the cell span information. The cell span represents the user roaming log in the cellular network (training set). Each row in the set tells the cell id where the user is; the time he did enter and leave this specific cell. Therefore, the whole set shows exactly the user movement during the survey period.

An important characteristic in any cellular network is that areas covered by base stations overlap, so that several cells may be seen in a single location. If overlapping cells have approximately equal signal strength, the phone may hop between cells even when the user is not moving (due to attenuation, reflection, shadowing and diffraction of the electromagnetic waves). In dense areas, this oscillation is significant. Therefore, there is no one-to-one correspondence between a physical location and the cell used by a phone. To overcome this problem, we cluster cells that tend to represent one physical location according to the following algorithm: All cells in the cluster are adjacent. The average length of a visit to the cluster is larger than the sum of the individual cells averages. Any proper subset of cells in a cluster does not satisfy the previous condition. The first condition simply requires that all cells in a cluster are near each other. The second condition tests oscillation: the average time spent visiting a cluster is larger than the sum of the individual times only when the user moves back and forth between the cells in a cluster. If the user is at a cell that belongs to multiple clusters it is unclear which of the clusters he really is at. For simplicity, we recursively combine all the clusters that have shared cells. We normalized the input and output vector so that values fall in the range [0; 1]. For example, we divide all cells data (current cell, next cell and cell history) by maximum cell number. In this way, we encoded the data in rational values. This encoding helps in faster and more accurate training. We want to predict where the user will be in any minute. Therefore, we need to transform the data to represent minute by minute activities. Neural Network inputs and outputs vectors have to be chosen carefully whether the model will be used in standard training or Bayesian analysis. Below is a description of the setup we have used. These values are based on previous research on models used in the same domain. Of course, we have tested several configurations and selected the best one based on prediction accuracy. Inputs to the network are: Cell ID represents the current cell in which the user is roaming. As described earlier, cells are clusters to overcome the problem of frequency hopping. Cells are also normalized to the range [0, 1], Cells history represents locations the user has been in. It is very important to note how history affects prediction accuracy. It encapsulates in a way the current pattern of the user, which helps predicting how he will move next. This value represents the hour when the user has entered the current cell. The value is normalized using the cosine function. This value represents the minute when the user has entered the current cell. The value is normalized using the cosine function, Day of week represents when the user has entered this specific cell. The value is according to the following scheme: The ID of the next cell the user will enter. As described earlier, cells are clusters to overcome the problem of frequency hopping. The following table is extract of the input/output vectors:

Cell ID	Cell History 1	Cell History 2	Cell History 3	Cell History 4	Cell History 5	Hr	Min	Day of Week	Next Cell
82	86	82	86	9	86	1	2	4	86
86	82	86	82	86	9	1	2	4	82
82	86	82	86	82	86	1	3	4	86
86	82	86	82	86	82	1	4	4	59

Table 1: Exact Input/Output Data Set

This model has one hidden layer (based on our research for previous models in the same domain). The number of hidden nodes will vary in the experiments in the range of 15 – 25 nodes. As described before, the Bayesian Neural Network model requires prior knowledge. It represents our initial belief before seeing the data. We define Gaussian distribution for network weights and biases. We also define hyper-parameters that are from conjugate inverse Gamma distribution. These hyper-parameters govern the possible values that the weights and biases may take instead of giving static values to them. Moreover, each relevant group of weights and biases (such as inputs to hidden weight vector) is given a separate hyper-parameter. In the same manner, prior structure for the residuals (noise model) is defined. We have divided part of the data that we have into training and testing data. We decided to use one month for training the model. The patterns used range from 26-7-2004 to 26-8-2004 and include all the parameters that we described in the input vector: Cell ID – Cell history – Start Hour – Start Minute – Day of week. The output vector consists of one value of Next Cell. The testing data that we use for prediction consists of one month patterns from 27-8-2004 till 20-9-2004. We define here Standard and Bayesian Neural Networks models used in our experiments: **Resilient Backpropagation:** Inputs: cell ID, 5 cell history, start hour, start minute, day of week, Hidden Nodes: 15 nodes**,** Output: next cell**. One Step Secant:** Inputs: cell ID, 5 cell history, start hour, start minute, day of week **,** Hidden Nodes: 15 nodes**,** Output: next cell. **Levenberg-Marquadt**: Inputs: cell ID, 5 cell history, start hour, start minute, day of week , Hidden Nodes: 15 nodes, Output: next cell. **Elam**: Inputs: cell ID, start hour, start minute, day of week , Hidden Nodes: 15 nodes, Output: next cell. **Bayes 1**: This is Bayesian Neural Network Model that has: Inputs: cell ID, 5 cell history, start hour, start minute, day of week , Hidden Nodes: 15 nodes, Output: next cell. **Bayes 2: Same with 25 hidden nodes. Bayes 3: same with no history**. **Bayes 4: same with 5 cells history and 15 hidden nodes**. **Bayes 5: same with 25 hidden nodes. Bayes 6: same with 15 hidden nodes.** We test our results against already established inference methods. We have chosen standard Neural Networks models as the benchmark. We use exactly the same Neural Network model and analysis the effect of the Bayesian Learning on the quality of the output solution. We also check speed and complexity of the models.

	Hidden nodes	**Training time**	**Epochs /**	**Cell History**	**Prediction Accuracy (exact)**	**Prediction Accuracy (Paging 6 neighbor cell)**
Bayes 1	15		3400	5	24%	35%
Bayes 2	25		2000	5	10%	16%
Bayes 3	15		2000	0	12%	24%
Bayes 4	15		1000	5	45%	64%
Bayes 5	25		600	5	47%	55%
Bayes 6	15		600	0	16%	39%
Resilient1	15	5 hrs	25000	5	0.5%	3%
Resilinet2	15	40hrs	250000	5	1%	5%
Levenberg-Marquadt	15	16 hrs	25000	5	0%	0%
One Step Secant	15		500	5	0%	0%
Elman / RP train	15	8 hrs	500	0	1%	4%

Table 2: Prediction Results

The above results show that Bayesian Neural Networks outperform Standard Neural Networks by 8% for worst case and by 30% for best case. Note that we compared the number of epochs to the number of samples generated. Note also that most of the Standard Neural Networks could not in the first place learn the user patterns and therefore could not predict the location. We test our model to a wider window. We generated test data for the periods from 30-10-2004 till 26-11-2004 and 26-11-2004 till 26-12-2004.

	Period	**Prediction Accuracy (exact)**
Bayes 1	1st Month	24%
	2nd Month	26%
	3rd Month	24%
Bayes 4	1st Month	40%
	2nd Month	48%
	3rd Month	36%

Table 3: Prediction Accuracy

Prediction does not work in every case because we sometimes just break from our daily routine. However, this break is still in most cases related to my location. We tried to see the outcome of searching nearby cells in case of the system hasn't found the user in the predicted cell. Next tables show results of prediction accuracy when we add paging neighbor cells according to:

	Period	Prediction Accuracy (exact)	Prediction Accuracy (paging 6 neighbor cells)
Bayes 1	1st Month	24%	35%
	2nd Month	26%	43%
	3rd Month	24%	40%
Bayes 4	1st Month	40%	55%
	2nd Month	48%	65%
	3rd Month	36%	54%

Table 4: Prediction Accuracy with Paging

These results are very promising. Prediction accuracy increases on average 60% in relative to checking only the one predicted cell suggested.

One model for each user might not be feasible for large deployment. In wireless networks with millions of users, this is obviously impractical. We decided to test if we can group similar people having the same daily activities into one model. In this experiment we grouped data from 5 MIT computer science students. We input this data into our model. We add one additional input that identifies the student. The below table shows results of the experiment:

User	Exact Prediction	Paging 6 neighbor cells
1	41%	60%
2	47%	70%
3	29%	46%
4	31%	50%
5	40%	59%
Average	37.6%	57%

Table 5: Cluster Prediction

Results show that we can actually group users with similar characteristics, maybe who are working in the same company, or have similar social activities. In this way, we can overcome the problem of having large number of users.

4 Conclusion

The main contribution of this work is a novel method for topology-independent user tracking. The expected wireless cost of tracking under the proposed method is significantly reduced, in comparison to the existing methods currently used in cellular networks. Furthermore, as opposed to other tracking methods, the worst case tracking cost is bounded from above and governed by the system, such that it outperforms the existing methods. The proposed strategy can be easily implemented, and it does not require a significant computational power from the mobile user device. Results show that prediction accuracy for this model outperforms all other discussed standard neural networks. Enhancing prediction accuracy by including cells geographical coverage and streets network can extend our work. Moreover, studying service prediction and its relation to location prediction is interesting feature that is currently being tackled by our team.

5 References

[1] Hassan Karimi and Xiong Liu. "A Predictive Location Model for Location-Based Services." GIS'03, November 7-8, 2003, New Orleans, Louisiana, USA.

[2] Alejandro Quintero. "A User Pattern Learning Strategy for Managing Users' Mobility in UMTS Networks" .IEEE Transactions on Mobile Computing, VOL. 4, NO. 6, November/December 2005.

[3]Jarno Vanhatalo and Aki Vehtari. "MCMC Methods for MLP-network and Gaussian Process and Stuff"– A documentation for Matlab Toolbox MCMCstuff. Laboratory of Computational Engineering, Helsinki University of Technology, 2006.

[4] Radford Neal. "Bayesian Methods for Machine Learning." NIPS Tutorial, 13 December 2004, University of Toronto.

[5] Jouko Lampinen and Aki Vehtari. "Bayesian Approach for Neural Networks – Review and Case Studies." Laboratory of Computational Engineering, Helsinki University of Technology,2006.

[6] Aki Vehtari, Simo Särkkä, and Jouko Lampinen. "On MCMC Sampling in Bayesian MLP Neural Networks." Laboratory of Computational Engineering, Helsinki University of Technology, 2006.

.