ACTIVE INFERENCE FOR PREDICTIVE MODELS OF SPATIO-TEMPORAL DOMAINS

BY

CANER KOMURLU

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science
in the Graduate College of the
Illinois Institute of Technology

Approved 
Advisor

Chicago, Illinois
May 2019

© Copyright by
CANER KOMURLU
May 2019

ACKNOWLEDGMENT

I would like to start with thanking my Ph.D. adviser, Dr. Mustafa Bilgic. He not only advised my research, but also guided me through this academic path. I learned so much from him, through discussions, collaborations and by his generous advice.

I thank my dissertation committee members, Dr. Aron Culotta, Dr. Boris Glavic, and Dr. Ankit Srivastava, for their contribution to my dissertation with their invaluable feedback and questions.

I would like to acknowledge National Science Foundation award no. IIS-1125412 from which I received funding for this research. Also, I would like to thank the Department of Computer Science at Illinois Institute of Technology for appointing me as a teaching assistant and supporting my research and education.

I thank the former and current members of Machine Learning Laboratory at IIT, Dr. Maria Eugenia Ramirez-Loaiza, Dr. Manali Sharma, Neil Getty, Jinjian Shao, Ping Liu, Jawahir Panchal, Anneke Soraya Hidayat, and Ruo Yang. I also thank my other collaborators outside Machine Learning Laboratory, Dr. Ali Cinar, Dr. Eric M. Brey, Dr. Judith Zawojewski, Dr. Hamidreza Mehdizadeh, Dr. Elif Seyma Bayrak, Dr. Banu Akar, Dr. Sami Somo, and Catherine Newman.

I cannot thank my wife, Ekin, enough, for being so patient, supportive, understanding, and cheerful throughout this long process. She has always been there when I needed.

Finally, I thank my parents, Günsel and Cenani, who always supported me from the very beginning, and instilled in me the enthusiasm to keep educating myself. I also thank my brother, Alper, for all his support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Active inference is the method of selective information gathering during prediction in order to increase a predictive machine learning model's prediction performance. Unlike active learning, active inference does not update the model, but rather provides the model with useful information during prediction to boost the prediction performance. To be able to work with active inference, a predictive model needs to exploit correlations among variables that need to be predicted. Then the model, while being provided with true values for some of the variables, can make more accurate predictions for the remaining variables.

In this dissertation, I propose active inference methods for predictive models of spatio-temporal domains. I formulate and investigate active inference in two different domains: tissue engineering and wireless sensor networks. I develop active inference for dynamic Bayesian networks (DBNs) and feed-forward neural networks (FFNNs).

First, I explore the effect of active inference in the tissue engineering domain. I design a dynamic Bayesian network (DBN) model for vascularization of a tissue development site. The DBN model predicts probabilities of blood vessel invasion in regional scale through time. Then utilizing spatio-temporal correlations between regions represented as variables in the DBN model, I develop an active inference technique to detect the optimal time to stop a wet lab experiment. The empirical study shows that the active inference is able to detect the optimal time and the results are coherent with domain simulations and lab experiments.

In the second phase of my research, I develop variance-based active inference techniques for dynamic Bayesian networks for the purpose of battery saving for wireless sensor networks (WSN). I propose the expected variance reduction active inference method to detect variables that reduce the overall variance the most. I first propose a DBN model of a WSN. I then compare the prediction performance of the

DBN with Gaussian processes and linear chain graphical models on three different WSN data using several baseline active inference methods. After showing that DBNs perform better than the baseline predictive models, I compare the performance of expected variance reduction active inference method with the performances of baseline methods on the DBN, and show the superiority of the expected variance reduction on the three WSN data sets.

Finally, to address the inference complexity and the limitation of representing linear correlations due to Gaussian assumption, I replace the DBN representation with a feed-forward neural network (FFNN) model. I first explore techniques to integrate observed values into predictions on neural networks. I adopt the input optimization technique. Finally, I discover two problems: model error and optimization overfitting. I show that the input optimization can mitigate the model error. Lastly, I propose a validation-based regularization approach to solve the overfitting problem.

CHAPTER 1

INTRODUCTION

Machine learning is a research field in *artificial intelligence* that solves problems by learning from experience. Typically, in predictive machine learning, the aim is to estimate the value of an attribute of a given instance based on patterns previously discovered from instances of the same domain. Among major paradigms of machine learning such as *unsupervised learning* and *reinforcement learning*, *supervised learning* encompasses problems in which preliminary data is used to adjust mathematical models which are then used for decision making. The key element that differentiates supervised learning from unsupervised learning is the fact that the data should originally have a target variable, namely *label*, available for every instance in the data set. That data set can then be used to adjust a mathematical model to map from input features to target variables [115]. This process of adjustment is called *training*, and it is indeed the process of finding optimal values for all parameters of the selected model that will make the model capable of estimating the labels of instances. Once this model is trained, it can then be used to predict labels of new instances [91].

A well-known application of supervised learning is spam filtering [149, 45]. In this problem, the goal is to detect whether an email is spam or legitimate by examining certain attributes of the email, such as the sender address, subject, email body, etc. A function is first trained on a number of emails that are previously labeled by human experts as spam or legitimate. Then, the same function is used to categorize new emails.

In a supervised learning problem, if the label values are categorical or qualitative, then the problem is called *classification*. Otherwise, if they are from a range or if

they are quantitative, then the problem is called *regression* [91, 45]. For example, in spam filtering, emails can be either legitimate or spam. As there are two categories, this is an example of binary classification. On the other hand, if, for instance, the problem requires to predict wind speed at a given time in a region, based on past weather information of various regions, then it would be regression as wind speed is a continuous value.

A prevailing impediment in supervised learning is that, for each instance, features can automatically be collected, yet the label needs to be manually set. For example, in spam filtering, attributes of an email such as words existing in its body, subject, sender address, date and time, etc. can automatically be gathered. On the other hand, whether it is spam or legitimate has to be detected by a human labeler. In the majority of the cases, labeling has to be done manually, thus it takes time and effort. In addition, in some specific problems such as medical diagnosis, labeling requires expert knowledge, making it costly.

Many paradigms have been proposed to overcome the problem of labeling costs. One approach is *active learning* [33, 120]. In pool-based active learning, we have plenty of unlabeled data, commonly represented as $\mathcal{U}$ and initially very little labeled data (or sometimes none at all), commonly represented as $\mathcal{L}$. Given a budget $\mathcal{B}$, we are allowed to select instances $(x_i, ?)$ from $\mathcal{U}$, obtain their labels from a human labeler (x_i, y_i), and add them to $\mathcal{L}$ ($\mathcal{L} := \mathcal{L} \cup \{x_i, y_i\}$) [101]. Since we have a limited budget to create the training set, we need to select those that are most informative and that accelerate the training of our predictive model. Many strategies have been proposed for detecting most informative instances such as uncertainty sampling [79], query by committee [37], and expected error reduction [111]. In uncertainty sampling [79], instances on which the predictive model is most uncertain are selected to be annotated. In query by committee [37], instances on which a set of predictive models

most disagree are selected. In expected error reduction [111], instances that are the most promising in reducing the error that the predictive model is expected to make, etc.

Let us consider the spam filtering problem in active learning perspective. We may have hundreds of thousands of emails in our data set, yet they are most likely not labeled as spam or legitimate. Labeling all of them will be very expensive. Therefore, we need to be selective. We randomly select a few of them to bootstrap a predictive model. We then select the next instance that will make the highest contribution to the training of the model. We label it and add it to our set of labeled emails, $\mathcal{L}$, which forms our training set. We then retrain our model with this incremented training set and select the next most useful instance. We keep iterating until we are out of budget or our predictive model performs well enough, whichever comes first.

1.1 Active Inference

In budget constrained supervised learning problems, active learning selects the most useful instances for training, as long as one has the freedom to choose instances to build up a training set. Similarly, in some problem definitions, one may have the budget to manually label instances at prediction time. In such cases, knowing the true label of some instances may help more accurately predict labels of the others. For such problems, a novel approach has recently been introduced to supervised learning, called *active inference* [15].

Active inference deals with the problem of selective gathering of information for some of the variables of a model with the objective of improving prediction for the remaining variables. Unlike active learning [120] which collects label information while training a statistical model, the main task here is to gather more information during inference to increase the predictive performance of the underlying model. The

underlying models tend to be graphical models or relational models, where observing the values of a subset of random variables helps with the prediction for the remaining variables. A few applications of active inference include node classification in which Bilgic and Getoor [16] query the labels of a few carefully chosen nodes in a network to let the underlying model (pairwise Markov random fields) condition on those labels to improve the prediction on the remaining nodes; and video analysis in which Chen et al. [28] manually analyze a few short segments of a video and let the underlying model (hidden Markov model) condition on the observed information to improve prediction on the remaining segments of the video. There are two main active inference categories: (i) feature acquisition, which gathers information on the input variables of the predictive model, and (ii) label acquisition, which gathers labels of some instances [13]. In this dissertation, we will focus on label acquisition.

In the context of the spam filtering example, for active inference to be applicable, the underlying model needs to be a collective classification model, where the labels of many emails are jointly predicted [119]. For example, assume that the emails form a network based on their senders and domains, and that the emails that are connected in this network are likely to have a similar label. Then, for a collective classification of these emails, active inference deals with the question of "given a limited budget, which emails should be manually labeled, so that the remaining ones can be predicted most accurately?".

1.2 Dynamic Bayesian Networks in the Context of Active Inference

In some problems, one may need to predict multiple variables at each prediction. Traditional predictive models, such as naïve Bayes, logistic regression, decision tree, etc. can only predict one variable. Therefore, usually in such cases, one will need to train a model per variable to be predicted. Unfortunately, this approach

ignores correlations potentially present between predicted variables. In the context of spam filtering, suppose one needs to predict whether an email is an advertisement along with whether it is spam. Labeling an email as an advertisement will probably affect the prediction of whether it is. However, aforementioned predictive models will need to predict these two labels independently.

Another challenging problem in predictive machine learning is to handle missing values. For example, naïve Bayes assumes that features are independent given class label [86]. However, when a feature value is missing, a naïve Bayes model will not be able to represent it in the calculation of the probability distribution of the label. Similar inconveniences apply to logistic regression as well as to decision tree.

Bayesian networks can easily overcome these two problems. They represent each feature as a random variable and explores conditional dependencies between them in the training phase. These dependencies are represented as a directed acyclic graph, which is also called the structure or the topology of the network [85]. The structure of the network is then used to extract conditional probability distributions or densities for the random variables.

Bayesian networks are powerful in representation. BNs can capture the joint probability of random variables compactly. By factorizing the joint probability distribution into multiplication of conditional probability distributions, BNs reduce the complexity of training from the product of the order of the number of values for each variable to the order of the summation of number of values for each variable. Therefore, the training phase becomes much more efficient as conditional probabilities require much less data than the joint probability. This capability is discussed in further detail in Chapter 2.

Dynamic Bayesian networks (DBNs), as an extension of Bayesian networks,

represent variables in a dynamically expending dimension [90, 61]. Without loss of generality, let this dimension be time. In such a scheme, the network has time slices each representing a time point in uniform intervals. Each variable is then represented once in each slice. Generally, dependencies between variables within a slice is replicated for each slice. In addition, new dependencies between variables are defined across time slices. Thereby, a variable's observation can affect probabilities of the same variable and its dependents in future time slices as well as past time slices. All these additional properties provide DBNs with the capability to model time involving physical phenomena. In our weather forecast example, we are interested in weather conditions every day. Conditions on each day are correlated to conditions on the previous day. Therefore there are temporal correlations which can be represented in DBNs.

Furthermore, if a domain involves dynamically expanding dimensions other than time, such as space, then DBNs can also deal with them as well. For example, in our weather forecast problem, suppose we are interested in not only a single region's weather conditions but many different regions. Clearly, these regions' weather conditions are correlated with one another. In such a case, for example, we can include variables for humidity, temperature and rain for each region and we can explore conditional dependencies among these variables across regions.

One problem with DBNs is that uncertainty of an inference on a BN is inversely correlated with the amount of observed values provided. Less evidence causes higher uncertainty in an inference, hence higher probability of prediction error. On a DBN, typically, the lack of evidence increases uncertainty as inference expands along the dynamic dimensions. For example, in weather forecast, if we provide all variables, such as humidity, temperature, wind speed, wind direction, etc., with evidence through observation on the first day of month and make predictions on the rest of

the month without any additional observation, then as we move forward towards the end of the month our predictions will be more and more uncertain.

1.3 Neural Networks in The Context of Active Inference

Dynamic Bayesian Networks have certain limitations. Although parameter learning is easy, optimal structure learning is exponentially hard. There are viable near-optimal methods for structure learning. Even then, exact inference is generally exponentially hard. Therefore, usually one might need to resort approximate inference methods. In certain circumstances, such as dynamic Bayesian networks where all variables are linear Gaussian, the exact inference is polynomial in the number of variables. However, this configuration imposes the limitation of linear correlations between variables.

Neural networks can be a promising alternative to DBNs, as inference on neural networks is fast. In addition, neural networks are strong in representing non-linear correlations, without compromising on the ease of inference. However, they are expensive to train: (i) it takes time and (ii) it requires ample data depending on the domain and the problem to be solved. These two problems can perhaps be worked out with today's computational and data gathering technologies. Meanwhile, there is one problem that needs to be addressed diligently in the context of active inference: For neural networks, integrating evidence into predictions is not straightforward. It requires additional methods to incorporate the evidence to predictions, whereas integration of evidence into prediction is a natural behavior of dynamic Bayesian networks. Therefore, while using neural networks as predictive models, one needs to resort some additional methods to enhance prediction performance using collected evidence.

1.4 Contribution of the dissertation

In this research, we developed active inference for (i) dynamic Bayesian networks and (ii) feed-forward neural networks to increase prediction performance at inference time by quantifying uncertainty with the applications in two different domains: tissue engineering, and wireless sensor networks.

Table 1.1. Application cases of active inference

Predictive Model	Tissue Engineering	Wireless Sensor Networks
Dynamic Bayesian Network	✓	✓
Feed-forward Neural Network	-	✓

1.4.1 Active Inference for Dynamic Bayesian Networks in Tissue Engineering to detect the optimal time to stop a wet experiment. In this scenario, we applied active inference for dynamic Bayesian networks in the domain of tissue engineering for the purpose of predicting vascularization and detecting the optimal time to stop a wet experiment. In tissue engineering, wet experiments take often weeks. In addition to long durations, they are costly. Terminating experiments earlier, observing the latest status and predicting about the finale with an acceptable uncertainty would help to get rid of unnecessary costs and time loss. Yet, detecting the right moment to stop an experiment depends on the experiment configuration. We proposed a DBN modeling for tissue engineering experiments. Then we applied active inference to detect the optimal time to stop an experiment, to collect evidence and to provide domain experts with reliable predictions on the end of the experiment. We discuss the details in Chapter 3.

1.4.2 Active Inference for Dynamic Bayesian Networks to detect optimal subsets in sequential prediction. In this problem description, instead of detecting the optimal time for a complete observation of the space, we focus on finding the optimal subset of observation among all candidate variables at each time of prediction. We investigated this problem in the wireless sensor network (WSN) domain.

For WSNs, battery consumption is a major problem that limits the lifespan of sensors. To mitigate this problem, instead of collecting information from all sensors, we collect from a subset and predict the rest. Therefore, there are two main tasks in this problem: (i) Building a predictive model to predict non-collected sensor readings using sensor readings collected at present and in the past, (ii) Finding the subset of sensors of which the readings at present will reduce the prediction error the most.

We modeled an entire WSN as a DBN in which we associated a Gaussian random variable with each sensor. We learned the structure as well as conditional dependencies from data. Then for each time being predicted, we actively selected sensors to be observed so as to minimize prediction error. We applied this approach on three different WSN datasets. We showed that, as a predictive model, our DBN performs better than the Gaussian process and the linear chain Bayesian model. We also showed that our active inference method outperforms the baseline active inference methods, the random selection and the sliding window selection. We discuss the details in Chapter 4.

1.4.3 Active Inference for Neural Networks to detect optimal subsets in sequential prediction. To address some limitations of DBNs in the previous domain, we replaced it with another predictive model, feed-forward neural networks. Then we applied active inference for feed-forward neural networks on battery saving for wireless sensor networks. The purpose is the same as the previous domain. In addition to the tasks described above, neural networks bear another task: how to incorporate sensor readings collected at present into predictions? Although DBNs naturally allow this evidence integration through inference, unfortunately neural networks do not. Therefore we proposed backpropagation of error to the input to match the predicted values of the observed sensors to their collected readings by adjusting all the inputs of the neural network, assuming that this will also reduce the error in

the prediction of non-collected readings. We first explored the origins of the prediction error of neural networks and showed that the error occurs due to two reasons: (i) model error and (ii) predicted values used at the input. We then showed that both errors can be reduced using input optimization by backpropagation of error. We finally discovered that input optimization can lead to overfitting. We proposed a regularization method using a validation set. We discuss the details in Chapter 5.

The rest of the thesis is organized as follows: Chapter 2 discusses the related work and the background of this research. Then, Chapter 3 presents our research on active inference for DBNs for modeling vascularization in tissue engineering. Chapter 4 illustrates our research of active inference for DBNs on an application of wireless sensor networks. Chapter 5 discusses our research on active inference for neural networks on an application of wireless sensor networks. Finally, I conclude our research and present potential future directions in Chapter 6.

CHAPTER 2

BACKGROUND AND RELATED WORK

In this section we provide our background along with related work for several topics that involve in this research: Bayesian networks and dynamic Bayesian networks, active learning, active inference, neural networks, neural networks for time series, tissue engineering, and finally battery optimization in wireless sensor networks.

2.1 On Bayesian Networks and Dynamic Bayesian Networks

Charniak describes Bayesian networks (BNs) (also known as belief networks, knowledge maps, probabilistic causal networks) as a research topic composed of collective effort put forth by many researchers [25]. Then he refers Pearl [97] for the name of Bayesian networks. It has been a very popular model in the artificial intelligence community, and it has been used in many different areas [25] such as medical diagnosis [57, 132], map learning [38], language understanding [27, 26, 48], computer vision [78], heuristic search [55].

First and foremost, BNs represent conditional independencies among variables and thereby alleviate parameter estimation complexity for joint probability distribution. For a glimpse of the difficulty, let us consider this case: Suppose we have n discrete variables, and for each variable, the domain sizes are $d_0, d_1, \ldots, d_{n-1}$, respectively. Then, the number of independent parameters in the joint probability distribution of these discrete values is $d_0 \times d_1 \times \cdots \times d_{n-1} - 1$. In the best case, all domains are binary, and the required number of parameters that need to be learned turn out to be $2^n - 1$, which is exponential in the number of variables. Even worse, data needed to learn parameters of the joint distribution will be equivalently large.

In such cases, Bayesian networks can drastically reduce the number of parameters to be learned by factorizing the joint distribution into a product of local distributions based on conditional independencies. Those local distributions are nothing but conditional distributions. Suppose that, a domain, $\mathcal{D}$, consists of variables, $v_0, v_1, \ldots, v_{n-1}$, and that each variable v_i can be conditioned on one variable, v_{i-1}. Then the joint probability distribution can be factorized into $P(v_0, v_1, \ldots, v_{n-1}) = P(v_0)P(v_1|v_0)\ldots, P(v_{n-1}|v_{n-2})$. Each factor here will require the following numbers of parameters to be learned respectively: $d_0 - 1, d_0 \times (d_1 - 1), \ldots, d_{n-2} \times (d_{n-1} - 1)$. Thereby the total number of parameters required will be: $d_0 - 1 + d_0 \times (d_1 - 1) + \cdots + d_{n-2} \times (d_{n-1} - 1)$. In contrast with joint probability distribution, this factorization requires the summation of these values which results the complexity in pseudopolynomial with the number of variables.

Furthermore, random variables may evolve over a dynamically expanding dimension such as time. Therefore, one may need a joint probability distribution that involves variables from different time stamps. Unfortunately such a distribution cannot be expressed unless number of time stamps is predefined. Dynamic Bayesian networks (DBNs) become even more handy regarding its capability to handle variable states across time, without loss of generality. Dynamic Bayesian networks are thoroughly discussed by many, such as Murphy [90], Ghahramani [46], Horsch [61]. DBNs are often selected due to their powerful expressiveness.

Some caveats require attention about DBNs. First, the structure of a DBN, i.e. the set of conditional independencies, needs to be carefully designed. On one hand, if too many conditional independencies are included, then the DBN will fail to represent the domain. On the other hand, if a handful of conditional independencies are included then the factorization nears the full joint probability distribution.

DBNs have been successfully applied to several real-world problems. A few

examples include managing water sources [19], gene regulatory networks [89, 99, 62], figure tracking [96], ranking [24], speech recognition [151], modeling environmental problems [140], and driverless cars [44].

Gaussian Bayesian networks are so far used in various contexts. Castillo et al. [22] designed a model for damage assessment in concrete structures of buildings using Gaussian Bayesian network modeling. They used their model in numerical propagation of uncertainty in incremental form and they also proved that conditional means and variances of nodes in their network are rational functions given evidence. Again Castillo et al. [23] used Gaussian Bayesian networks to model traffic flow.

2.2 On Active Learning

Active learning, after being introduced to supervised learning in mid-1990's, attracted vast attention and ample research has been carried out in active learning.

Bilgic and Getoor focused on uncertainty sampling and pointed out at its weakness about selecting outliers. They proposed a three-step solution in network data: (i) exploration by randomly sampling instances, (ii) expansion by choosing a random neighbor of instances, and finally (iii) connection by picking more instances by focusing on those that have maximum neighbor selected previously [14].

In the same line of work, Bilgic et al. applied active learning on network data. They asserted that links in a network where nodes are to be classified, can be utilized. Furthermore, they claim that active learning can be improved by using links as features for instances. As the number of neighbors vary for each instance, they aggregate connections into one feature. They use two classification models. One considers only the original features, the other considers in addition the aggregated feature. Based on their disagreements, the active learning algorithm picks the most uncertain instances for training [18].

In a different line of work, Ramirez-Loaiza et al. studied the performance change of active learning in various combinations of active learning approaches with classification models, under different performance measures. They showed that the choice of the classification model is as important as the active learning approach to increase the performance. They also showed that an active learning approach selected to increase a performance measure can harm the performance with respect to other measures [104].

In active learning, Ramirez-Loaiza et al. introduced the concept of interrupting the expert while the expert is providing the active learner with annotation. They formulated the cost of annotation as time spent by the expert. In text classification, they empirically showed that some interruption is always better than none [102, 101]. In their follow-up work, they extended their study with user study and proved that an interruption after the first 25 words has shown a much better performance than allowing the expert to read the first 100 words. Note that, early interruption is prorated to the cost of annotation and therefore reducing cost per annotation helped the active learning to have more instances annotated [103, 101].

Sharma and Bilgic focused on uncertainty sampling of active learning. They split the uncertainty into two categories: (i) uncertainty by lack of evidence for any class, and (ii) uncertainty by contradicting evidences for multiple classes. They investigated this separation on naïve Bayes, logistic regression, and support vector machine [122, 124, 121].

Sharma et al., similar to the work by Ramirez-Loaiza et al., tried to improve active learning by requesting rationales from the human annotator for each instance labeled. They integrated the rationale to the instance as a new feature. They showed that this new method has increased the performance of active learning independently from the classification model [126, 125, 121].

In a follow-up work, Sharma and Bilgic extended the rationales approach by providing explanations to the active learner. They asked the human expert to highlight phrases supporting the selected label and those contradicting the selected label. They empirically showed on three text datasets that teh active learning with explanations outperforms the active learning with rationales, the traditional active learning [123, 121].

Additionally, Sharma et al. applied using rationales to detecting operationally significant anomalies in commercial flights. They first used unsupervised learning techniques to detect anomalies. Then, using active learning with rationales, they labeled few instances as operationally significant or insignificant, integrating rationales by subject matter experts. They showed that their method reached 75% improvement over the state-of-the-art [127, 121].

2.3 On Active Inference

Information acquisition is examined under two categories in Supervised Learning: *feature acquisition* and *label acquisition*. The first category includes various methods that deal with the problem of deciding which features of instances to gather in cases where features are not readily available and/or expensive. Methods in label acquisition concentrate on the problem of how to collect labels effectively. This collection can be practiced during learning as well as during prediction, depending on objectives and constraints. Addressing effective strategies for label acquisition in the learning phase are collected under the title of *Active Learning*. Active Learning pursue objectives of training a supervised learning model with a smaller budget, or fewer instances in other words. In real world, data is often structured and instances are not independent. Therefore, true label information of an instance can help predicting labels of some others. Based on this presumption, label acquisition is addressed also during prediction to mitigate prediction error [13]. The the question emerges as *which*

instances should be selected for true labeling to reduce prediction errors? Strategies designed to answer this question are called as *Active Inference*. Active Inference name was first proposed by Rattigan et al. in the context of network data [107].

Active inference [15] is a technique of selective information gathering during inference/prediction with the objective of maximizing prediction accuracy. Though active inference and active learning sound similar, the key setting and objective are different. Active learning's [120] objective is to train a predictive model with as little supervision as possible. To achieve this, the system queries an oracle/expert for only the most informative objects. Active inference, on the other hand, assumes that there is an already-trained predictive model that has been deployed, and additional information can be gathered at inference time to help the predictive model make more accurate decisions.

For example, in a medical diagnosis setting, a predictive model that has already been deployed can gather additional information about a patient (for e.g., the system can order new laboratory tests) to make a more informed decision. Like active learning, active inference has a limited budget (money, time, risks, etc.) and has to be selective about what information it chooses to gather. Unlike active learning, however, active inference already has access to an already-trained model, and hence active inference's objective is not to make the model better, but rather, it is to gather additional information to condition on so as to make the predictions more accurate.

Active inference is even more applicable to domains where several predictions need to be made collectively and the predictions are interrelated. For example, when one is classifying frames of a video footage for whether a given suspect is present or not, one can learn and utilize the spatio-temporal correlations in the video frames. Given a trained model, active inference can then collaborate with a human user, asking the user to inspect certain frames, and the system can condition on the infor-

mation provided by the user to make predictions on the rest of the frames. To be able to condition the predictions on the information provided by the user, the underlying model needs to be a joint model, such as a Bayesian network.

Active inference was previously applied by Bilgic and Getoor ([15, 16]) for general graphs. They used active inference to query the labels of a small number of carefully chosen nodes in a network to condition the underlying model on these collected labels to increase prediction performance on the remaining nodes. In [16], they used iterative classification algorithm as the underlying predictive model, whereas in [15], they used pairwise Markov random fields as the underlying predictive model.

Active inference was also used by Chen et al. [29] on hidden Markov models, to analyze short chunks of a video on which the underlying model can condition and make better predictions on the remaining segments. Chen et al. [28] used again active inference for video analysis which was modeled as general graphical models. Active inference was also discussed in the context of hidden Markov models by Krause and Guestrin [74]. Finally Krause and Guestrin [72, 73, 74], and Bilgic and Getoor [17] formulated active inference as optimal value of information on graphical models.

2.4 Neural Networks

Artificial neural networks, or simply neural networks (NNs) in the field of machine learning, are human brain-inspired mathematical models designed for prediction purposes [4, 115, 56]. They can be used for supervised learning purposes as well as unsupervised learning purposes [91]. A neural network is composed of neural units or perceptrons.

Perceptrons are first introduced by Rosenblatt [110] in 1958. Each perceptron has multiple inputs, $\boldsymbol{x}$ and one output, y.

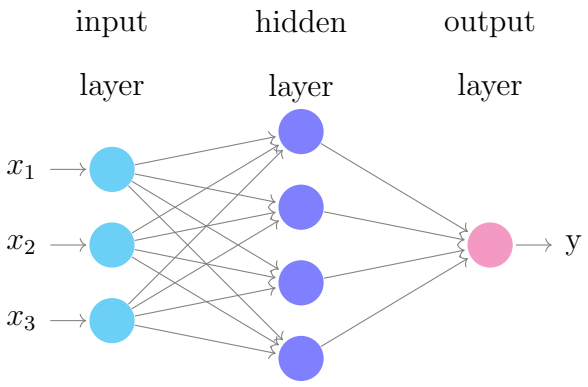


Figure 2.1. An example of multilayer perceptron

$$y = f(\boldsymbol{x}) = a(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{b}) \tag{2.1}$$

In Equation 2.1, $\boldsymbol{w}$ is the weight vector, $\boldsymbol{b}$ is the bias vector, and a is an activation function. The activation function can be a variety of mathematical functions, such as rectified linear unit (ReLU), logistic sigmoid, hyperbolic tangent, step, identity, radial basis function, etc. [128, 66].

A neural network can be composed of multiple layers of perceptrons such as in Figure 2.1. These type of networks are named as *multilayer perceptrons*(MLP), also as *feed-forward neural networks*, or as *deep feed-forward networks* [49]. Every layer has one or more perceptrons except the input layer. Typically, all perceptrons in one layer have the same activation function, yet their input weights are different, which makes them different decision-makers. Every perceptron behaves like a decision-making agent based on the input signal.

The input signal originated from multiple sources are weighted with a set of coefficients and added some bias values. Then an activation function is applied. If the input signal is strong enough, then the perceptron is activated and generates a signal at the output, otherwise inhibited and does not generate a signal. Depending

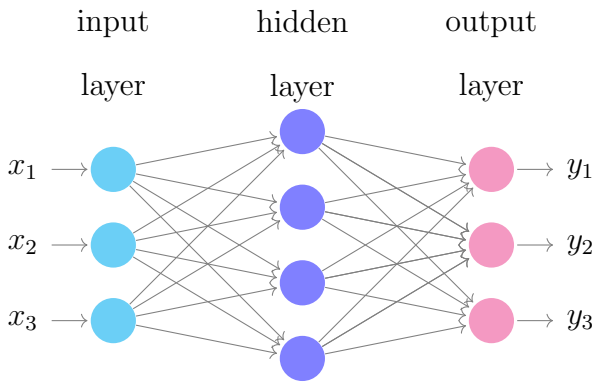


Figure 2.2. A sample of multilayer perceptron with multiple output

on the domain and the problem needs, this activation behavior can be modified. The step function output is defined on $\{0, 1\}$. Logistic sigmoid function generates an output in the range of $[0, 1]$ as it is a continuous function. Hyperbolic tangent is a continuous function like logistic sigmoid, yet it generates an output in the range of $[-1, 1]$. Finally, ReLU permits the weighted sum of the input signal to the output if it is positive.

One needs to be careful about neural network design, depending on the problem definition. If the objective is classification, the output layer can have the step activation function. However, if one needs to have a probabilistic behavior, logistic sigmoid is also an option as it generates values in the range of $[0, 1]$. When the objective is regression, then the output layer may not use the step function, nor logistic sigmoid. Generally identity function is used.

Similar to DBNs, NNs have the advantage of supporting multiple label, i.e. multiple output prediction (e.g. Figure 2.2). This helps the model reuse the patterns and correlations learned from the domain in predicting all output. Alternatively, on the same domain, one can train a single output model for each variable to predict, such as a logistic regression or a decision tree, yet predictions will not support correlations between variables to predict.

Like any other machine learning model, neural networks need training before they are deployed. Generally, gradient descent or its variants [112] (e.g. Nesterov accelerated gradient [94], Adagrad [42], Adadelta [147], RMSProp[1], Adam [68], Adamax [68], Nadam [41]) are used to train neural networks. A very common variant of gradient descent is stochastic gradient descent (SGD). The objective of gradient descent is to keep updating every parameter of the model until its contribution to the error is minimized, given a training set.

$$w_i^+ = w_i + \eta \frac{\partial \operatorname{Err}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial w_i} \tag{2.2}$$

Figure 2.2 outlines the update of a parameter with respect to the error of the model on a training set. w_i is a parameter in the model. $\operatorname{Err}(\boldsymbol{x}, \hat{\boldsymbol{x}})$ is the error, also known as loss, if the model on the training set. $\boldsymbol{y}$ is the vector true labels of the entire set, or a subsample of it. $\hat{\boldsymbol{y}}$ is the vector of predicted labels for the instances of $\boldsymbol{y}$. The gradient of error with respect to the current value of the parameter, $\frac{\partial \operatorname{Err}(\boldsymbol{y}, \hat{\boldsymbol{y}})}{\partial w_i}$ is calculated, then used for dowsing its global minimum. η is the step size. While the gradient provides with the direction and the length of the step towards the next value, η adjusts the size of the step. Finally w_i is updated to w_i^+ as the next value. This update iterates until the error is minimized. In the context of feed-forward neural networks this optimization is called *backpropagation*, as it calculates the gradient of the error and propagates it back to all parameters [114].

For temporal or sequential domains, an extension of feed-forward networks is proposed: recurrent neural networks (RNNs) [47, 98, 108, 109, 113]. As in Figure 2.3, RNNs are composed of one or more hidden layers. One layer has recurrent connec-

[1]This is not a published method, rather a discussion in lecture notes by Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky in Coursera, *Neural Networks for Machine Learning Lecture 6a: Overview of minibatch gradientdescent*

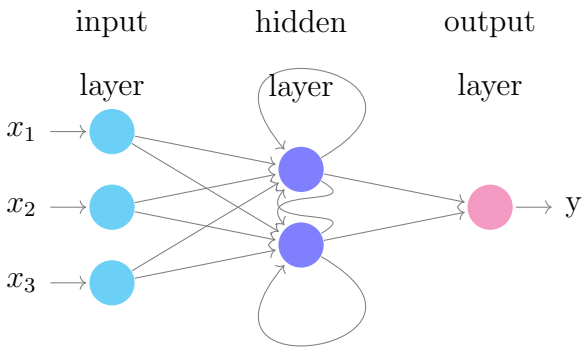


Figure 2.3. Typical topology of a recurrent neural network

tions. That is, the input to this layer is composed of the output of the previous layer and its own output. The values from itself are generated in previous forward pass, i.e. inference. Equation 2.3 shows the equation of the output of the hidden layer. $\boldsymbol{h}^+$ is the output of the recurrent layer at the current inference, while $\boldsymbol{h}$ is the output of the recurrent layer at the previous inference, and $\boldsymbol{u}$ is the weight matrix of $\boldsymbol{h}$. $\boldsymbol{x}$ is the output of the layer preceding the recurrent layer, or the input of the network, $\boldsymbol{w}$ is the weight matrix of $\boldsymbol{x}$, and $\boldsymbol{b}$ is the bias vector. $\boldsymbol{h}$ is also called *internal state* of the neural network as it is stored until the next inference, and used in it.

$$\boldsymbol{h}^+ = f(\boldsymbol{x}) = a(\boldsymbol{w} \cdot \boldsymbol{x} + \boldsymbol{u} \cdot \boldsymbol{h} + \boldsymbol{b}) \tag{2.3}$$

To train an RNN, backpropagation algorithm is utilized and extended. The extended version is called backpropagation through time (BPTT) [142, 143, 144]. BPTT simply propagates the gradient of the error backwards in time over $\boldsymbol{h}$ which is the output of the hidden layer in the previous time inference.

Cho et al. used RNN for machine translation. They built two RNN sequences. One for encoding the other for decoding. The first encodes the input sequence into a fixed size vector representation. Then the decoder RNN, initialized with the encoded

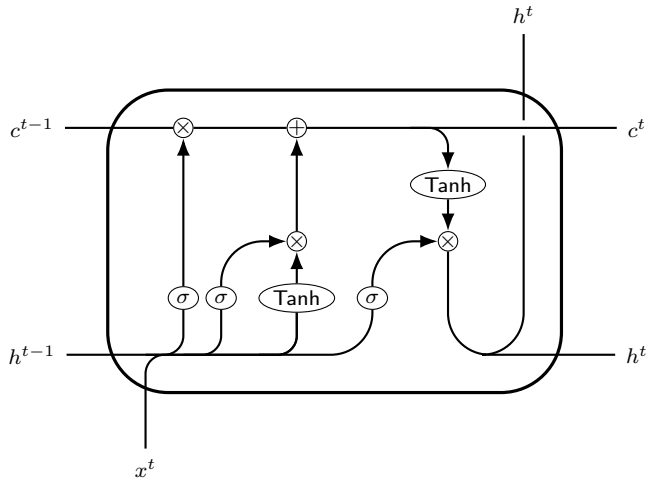


Figure 2.4. Typical structure of a long short-term memory cell

vector generates the translated word sequence [32].

Bahdanau et al. [6] addressed some limitations in fixed length vector representation in encoder-decoder approach for neural machine translation, proposed by Cho et al. [32] and Sutskever et al. [134]. They claim that fixed vector representation pose problems in representation especially when the input sentence is long. Therefore, [6] propose an extension to the encoder-decoder model that learns to align and translate jointly. They use a bidirectional RNN model to select more important words in translation for each output word [6].

Inspired by neural machine translation approach proposed by Bengio et al. [10], Ranzato et al. use RNNs to predict the next frame in a video given all past frames. They augment the RNN with convolutional layers placed not only at both the input and output of the recurrent layer but also at the transfer of the hidden state from time t to time $t + 1$. They tested their model in UCF-10 dataset [131] for human activity recognition [105].

As investigated by Hochreiter and Schmidhuber, by Hochreiter et al., and then by Pascanu et al. one major problem with RNNs is the difficulty of training.

Although BPTT is very promising in theory, gradients can easily explode or vanish over long sequences. This is a major impediment to updates by SGD, unfortunately resulting no learning of long-term dependencies [59, 60, 95]. For instance, in machine translation, a decision at the last time step of a word sequence composed of 30 words may very well depend on the initial word. Therefore, in training, the error gradient needs to be propagated 30 steps back which can easily vanish the gradient or explode it. Such cases are called long-term dependencies. To support learning long-term dependencies, Hochreiter and Schmidhuber proposed a new topology, long short-term memory (LSTM).

Figure 2.4 shows a typical LSTM cell which is a sophisticated version of an RNN's recurrent hidden layer. The structure is deliberately complicated to sustain long-term dependencies. In this structure, there are two hidden states, h^t and c^t. h^t is initialized to represent the stored information that is generated by the current time inference using the previous internal state, h^{t-1} and the current input to the network, x^t. Meanwhile, c^t works as a registry for how much the previous internal state should be forgotten and how much the new information should be stored in the current internal state. The first multiplication defines how much the cell will forget on the previous state. The following summation defines how much it adds up the new information in the internal state.

LSTMs are used for many different problem domains that require to support long-term dependency. Some notable studies are phoneme classification [52, 53], handwriting recognition [54], sequence generation and machine translation [51, 134], speech recognition[117, 116], anomaly detection [83], weather forecast [145], etc.

Graves and Schmidhuber compared bidirectional LSTMs with other recurrent and regular topologies, unidirectional LSTMs, bidirectional RNNs, unidirectional RNNs, and finally MLPs for the problem of framewise phoneme classification. Though

the number of hidden units/cells varied for each topology, they used the same training algorithm, BPTT and training parameters. They claimed that bidirectional models outperformed unidirectionals and MLPs. In addition, LSTMs were faster in training and more accurate than RNNs [52].

In [53], Graves et al. improved the framewise phoneme prediction using bidirectional LSTMs by coupling them with hidden Markov models (HMMs). In [54], Graves et al. used LSTMs for unconstrained handwriting recognition. In [51], Graves used LSTMs to generate sequences. The problems that he dealt were text prediction, handwriting prediction, and handwriting synthesis.

Donahue et al. used LSTMs in the video domain. They tackled two problems: activity tracking and text generating. They stacked one layer of LSTM on top of the other, and they fed the first LSTM layer with the output of a convolution layer. They treated the output of the second LSTM layer as their predictions Donahue et al..

Xingjian et al. used LSTMs for precipitation prediction given atmospheric radar data. Their data was composed of radar images that showed precipitation level at each cell of a grid at each time point. Their architecture was composed of two separate sequences: (i) Encoder, and (ii) Predictor. Each sequence was composed of two stacked LSTMs. That is, one LSTM layer was placed on top of the other. Both layers had convolution layers in their input. They also transmitted hidden and cell states through convolution layer from one time step to the next [145].

Srivastava et al. used LSTMs in supervised learning approach to encode video frames into a fixed length representation similar to Xingjian et al. [145], Sutskever et al. [134], Bahdanau et al. [6]. However, instead of generating a sequence, Srivastava et al. they predicted only the next time step [133].

2.5 Neural Networks on Time Series

Adya and Collopy reviewed prominent published studies on neural networks used for forecasting. They made a systematic evaluation of 48 different works. They explored the quality and impact of each of these works mainly in two perspectives: (i) Quality of implementation (ii) Quality of validation. They concluded that 21 of them had problems in validating their work, 16 had problems with implementing their models, 11 had good quality of implementation, as well as of validation. Out of 48 studies, 30 concluded that neural network models are better their contemporary state-of-the-art models, 13 studies were inconclusive whether neural networks have been better, and seven did not make a comparison [1].

Following Adya and Collopy's study [1], Zhang et al. also reviewed literature for forecasting in time series domains using neural networks. They first introduced the difficulty of forecasting in time series domains originated from non-linearity between input and output variables. The first advantage of neural networks for forecasting is that neural networks are data-driven self-adaptive models. That is, they can learn the domain with very little to non domain knowledge. Second advantage that they discussed is that the capability of neural network in generalizing. Third, they pointed out to the non-linear representation capability of neural networks. They continued their discussion with a brief description of neural networks. Then they presented some different application domains of neural networks for forecasting. Finally, they concluded with a discussion of potential issues of neural networks such as defining the number of hidden layers, and neural units in each layer, as well as normalizing the input, alternative to conventional backpropagation [148].

Gorr et al. proposed a densely connected single hidden layer neural network equipped with a linear decision rule administered by domain experts to predict student GAPs in a professional school, then they compared their model with linear regression and stepwise polynomial regression. They concluded that the comparison results were

not statistically significant [50].

Ho et al. used multilayer perceptrons for short-term power load forecasting. Also they proposed a new learning algorithm for neural networks, *Adaptive Learning Algorithm*, which adjusts the learning step, η, and the momentum, α based on the gradient of the error and the change on the parameter values. They compared their learning algorithm with backpropagation using gradient descent, and claimed that the proposed learning algorithm yields a faster learning [58].

Applying recurrent neural networks on time series data dates back as early as 1991 [34, 35, 82, 36, 75]. Logar, in her PhD thesis, compared recurrent neural networks against feed-forward neural networks and ARMA (autoregressive moving average) models in several different time series domains. She also proposed an improvement to gradient decent algorithm by projecting the weights to a potential trajectory and eliminating many steps, after the assumption that weight trajectories through gradient decent search are locally parabolic [82].

In addition to Logar, Kuan and Liu also compared recurrent networks with feed-forward networks. They conducted an empirical study on foreign exchange rate data. They proposed an automated two-step parameter learning approach. Then they experimented various topologies of feed-forward neural networks and recurrent neural networks on different currency exchange rates such as, Canadian dollar, British Pound, German mark, Japanese yen, and Swiss franc. According to their results, there was no decisive superiority between feed-forward networks and recurrent networks. Each outperformed the other in different cases, unless they tied [75].

Connor and Atlas showed that feed-forward neural networks are non-linear autoregressive (NAR) models. They also showed that recurrent neural networks are non-linear autoregressive moving average (NARMA) models. Then on a power system

regional load forecasting problem, they showed that RNNs with the help of feedback resulted better than feed-forward NNs [34].

Connor et al., in their follow up work, confirmed that RNNs are a type of NARMA. Based on their empirical studies on electric load forecasting, they showed that RNNs can be superior to feed-forward NNs in state dependent domains or domains having trends [35].

In their other follow-up work, Connor et al. showed that feed-forward NNs are a special case of NAR, and recurrent neural networks are a special case of NARMA. Therefore, similar to the advantage of NARMA over NAR in time series domain involving a state transition behavior, RNNs are superior to feed-forward NNs in certain time series domains [36].

Malhotra et al. used stacked LSTMs for anomaly detection in time series. They trained LSTMs first on non-anomalous trainin data. In prediction, they used the model to generate the data. Then they fit a multivariate Gaussian distribution to the generated data. Finally they used the distribution to decide on anomalous cases [83].

Sutskever et al. used LSTMs for sequence to sequence learning and machine translation. Their objective was to translate an English sentence to a French sentence. They trained two distinct LSTM models. First they encoded the input sentence into an internal state. Then the second model, initialized with the encoded internal state is used to generate the target sentence [134].

2.6 On Tissue Engineering

Although the human body has a great capability to heal, sometimes the tissue loss is so severe that the body cannot heal completely. For such cases, the missing tissue is replaced with autologous tissue or donor tissue. These approaches are limited,

and tissue engineering offers a potential alternative of growing new tissue using cells on biomaterial scaffolds [12, 80, 84]. The process starts with a small number of cells seeded into a biomaterial scaffold, the structure that holds the newly formed tissue together. If stem cells are used, they differentiate into specific tissue cells and the tissue cells multiply to fill the space. At the same time, blood vessels from surrounding tissue grow into the scaffold, carrying oxygen and nutrients to the cells and removing carbon-dioxide and other waste from them. The scaffold that holds the tissue cells degrades over time, allowing the cellular matrix to develop with the blood vessels to generate a fully functional tissue.

In our research, we focus on the vascularization aspect of the tissue engineering process, which is illustrated in Figure 2.5. When a tissue cell is far from the service range of a blood vessel, it cannot receive the oxygen and nutrients it needs; hence, it enters into a "distressed" phase, called *hypoxia*. Such a cell signals its distress by releasing solubale chemical signals such as vascular endothelial growth factor (VEGF). VEGF diffuses into the region and upon binding the outer layer of a blood vessel, this chemical initiates the process of sprouting a new blood vessel. The tip of the new blood vessel sprout, called the `Tip Cell`, grows stochastically in the direction of the VEGF. `Stalk Cells` proliferate behind the `Tip Cell` forming a new blood vessel as it elongates. When two different blood vessel sprouts meet, they connect (*anastomosis*) and the blood circulation starts through the newly formed blood vessel loop. If a cell stays in hypoxia for an extended amount of time, it dies. Therefore vascularization is crucial for healthy tissue growth.

The vascularization process is both a temporal and stochastic process. Over time, the tip of the blood vessel elongates stochastically in the direction of the VEGF gradient, searching for the distressed cell. The direction that the blood vessels grow through is stochastic because i) the blood vessels tend to do some exploration in their

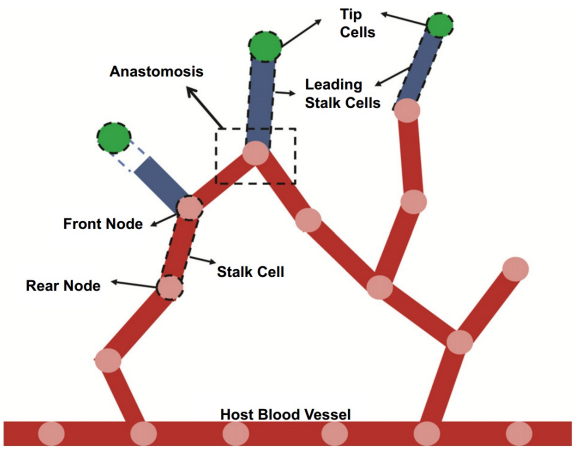


Figure 2.5. Illustration of vascularization, including `Tip Cell`, `Stalk Cell`, and anastomosis. Figure courtesy of Mehdizadeh et al. [88].

search for the distressed cell, ii) scaffold structures can sterically hinder movement of the cells up the gradient, and iii) there are other unaccounted and unknown factors besides VEGF that affect vascularization. Because the process is both temporal and stochastic, we model the process using a dynamic Bayesian network, which we explain in detail in Chapter 3.

Tissue engineering is an active research field where a number of papers tackle different aspects of the problem using in-vivo and in-vitro laboratory experiments (e.g., [2, 30, 31, 63, 64, 146]). There have been also computational models of tissue development, such as multi-agent systems (e.g., [5, 7, 11, 20, 88]). Artel et al. [5] proposed a multi-agent model to simulate vascularization in polymer scaffolds. They compared effects of various growth factor concentrations and scaffold porosities on vascularization. Mehdizadeh et al. [88] observed the contribution of porosity and interconnectivity of scaffold to vascularization. Bentley et al. [11] also proposed an agent-based model for vascularization in which they observe the effect of different growth factor conditions on blood vessel growth. Bailey et al. [7] used an agent-based model to represent an existing blood vessel network and they simulate white blood vessel trafficking. Finally, Byrne et al. [20] modeled tissue differentiation using

agent-based models under different settings of scaffold porosity and degradation rate.

2.7 On Wireless Sensor Networks and Battery Optimization

Wireless sensor networks (WSN) have become more and more common in the automated world. In an abundant variety of deployment, WSNs are used for many sensing purposes such as measuring temperature, humidity, light; sensing motion, vibration; detecting rain, fire; surveying traffic, etc.

Key elements of WSNs are sensors which are small-scale agents equipped with sensing devices, a processing unit, communication components, and an energy source. In majority of cases, sensors rely on their battery for energy as they are mechanically independent. The lifespan of each sensor is determined by its battery capacity and rate of energy consumption. Many solutions are proposed to increase battery life such as increasing battery capacity or reducing device demand. Many other strategies have been proposed in addition to hardware optimization [3].

A sensor node performs three essential tasks: i) *sensing* in which it converts a physical quantity into a representation, ii) *processing* in which it treats and saves readings, and iii) *communicating* in which it sends and receives data packages. In terms of energy consumption, sensing and processing are negligible compared to communication.

Many studies have addressed predictive models for energy saving on wireless sensor networks (WSN). Elnahrawy and Nath [43] propose prediction by context awareness. They discretize readings and use naïve Bayes classifier to predict these discrete values. They use the geographical topology of WSN as the basis of node neighborhood in their model.

In a similar way, Wang and Deshpande [141] clustered sensors and discretized readings. Then they applied compression techniques on descretized data to detect

correlations between sensors within each cluster. Using compression-based correlations, they fit joint distributions for variables in each cluster, and they incorporated the joint distributions into a probabilistic model, which they referred as decomposable model, to predict some sensors instead of reading them.

Deshpande et al. [39] also applied a model based predictive approach to saving energy. They trained a multivariate Gaussian distribution which models a wireless sensor network and let user run queries on their model with arbitrary confidence intervals. Depending on target query and confidence interval, their model decided which sensors to observe. Their approach involved the entire set of sensors as one multivariate distribution and they incorporated only observations of the prediction time. In our case, we make use of current observations and all past observations.

Another approach to energy optimization is scheduling sensors for reading. Slijepcevic and Potkonjak [129] addressed this approach to seek an efficient way of clustering sensors so that each cluster alone covered the entire area of surveillance, and that clusters alternated for sensing one at a time. They turned this problem into an area cover problem and maximized the count of clusters so that they could keep as many sensors silent as possible at each turn.

2.8 Examples of Applications in Spatio-temporal Domains

One spatio-temporal domain on which many applications of prediction have been conducted is activity recognition. In this domain, usually, measurements related to human activity are collected for prediction purposes. The data is collected over time. The temporal data is considered as features. Using these features, activities are predicted.

In one such work, Bao and Intille predicted a person's activities using readings collected from five biaxial accelerometers attached to that person's body (one on

biceps, one on a wrist, one on the waist, one on an ankle and one on a thigh). They collected acceleration values within the range of ±10 G from each accelerometer with a frequency of 76.25 Hz (every 20 ms). Using these readings, they predicted the human activity every 6.7 s with a time window of 512 samples window. Every activity is categorized into one of the 20 activities: walking, sitting & relaxing, standing still, watching TV, running, stretching, rubbing, folding laundry, brushing teeth, riding elevator, walking carrying items, working on computer, eating or drinking, reading, bicycling, strength-training, vacuuming, lying-down & relaxing, climbing stairs, and riding escalator. For predictions, they used decision trees and obtained an overall accuracy of 84% [8].

In a similar work, Kwapisz et al. focused on the same problem using only cell phone accelerometers carried on human body. Among many other sensors included in cell phones or portable music devices, such as optical sensors, audio sensors, GPS sensors, they only used accelerometers. These accelerometers are, unlike those worn on body in the work of Bao and Intille [8], are not biaxial but triaxial. In addition to two axes acceleration measurement, they also have gravity direction which introduces a new feature that helps discriminating activities. In their configuration, each person carried one device. They converted 10-seconds time windows, which contained approximately 200 readings, into an example. For each example, they generated features for each axis: average acceleration, standard deviation of acceleration, average absolute difference, average resultant acceleration, time between peaks, and binned distribution. As classification models, they used decision trees (J48, an extension of C4.5 [100]), logistic regression, and multilayer perceptrons. They defined six activities, walking, jogging, upstairs, downstairs, sitting, and standing. In overall accurracies, they could reach 91.7% using multilayer perceptrons, 85.1% using J48, and 78.1% using logistic regression [76].

Zhou and Hu investigated different techniques of motion tracking for rehabilitation purposes, in terms of instruments. They first split them into three categories: non-visual tracking, visual tracking, and robot-aided tracking. They then scrutinized each technique in these categories, by assessing advantages and disadvantages. Under non-visual tracking techniques, they listed inertia-based methods which are also known as accelerometers, magnetic-based techniques, glove-based techniques and others. Glove-based techniques are used to track hand gestures and to reconstruct motor functions. The category of others include for example acoustic devices. They claim that though acoustic devices are highly used in the medical field, for motion tracking of human body, they are not very suitable as those devices are rather large and require large surface to reflect acoustic waves. Under visual tracking systems, they list three sub-categories: visual marker-based tracking techniques, marker-free visual tracking techniques, and combinatorial techniques. In marker-based techniques, the subject person carries some light reflectors (in the passive case) or light sources on different spots of her body. Then using cameras placed in different angles, the subject's motions are tracked. In marker-free cases, the same task is tried to be achieved without using actual devices mounted on the subject's body. Finally, in robot-aided category, they discuss various techniques in which robotic systems are actively used to treat patients [150].

Motion tracking in video is another interesting machine learning application for spatio-temporal domains. For example, Klaser et al. used local descriptors composed of 3D gradients to track motion in video. Given a video, they first scanned it to determined local regions using an interest point detector or by dense sampling of the video. The sampled regions usually come with coordinates that describe the extent of their neighborhoods. Each neighborhood, also known as an interest region is then converted to a feature vector. Finally, the whole video is represented by a set of such feature vectors. Similar to the bag-of-words approach, they used these features to

classify motions in a video into certain predefined categories. For example, in the Hollywood actions dataset [77], they classified motions in videos into eight different actions: answering phone, getting out of the car, hand shaking, hugging kissing, sitting down, sitting up, and standing up [69].

Another method of motion tracking in videos is temporal difference, proposed by C. H. Anderson. In this method, the difference between consecutive frames is calculated. All regions that exceed a certain threshold in terms of difference are considered as motion regions. In these regions, not only the moving object but also the background that was occluded in the previous frame but then appeared in the next is detected as moving region. The background is eliminated by calculating the motion vector and filtering only the vectors that exist for a number of consecutive frames. Finally, only the moving objects remain [21]. Lipton et al. pointed out to a common problem with this method: it tends to combine the background region that yet has been left into the moving object. They also discussed certain weaknesses of this method such as, if there is a significant camera motion, the method simply does not work, or if the object is occluded then this method has difficulties to keep track of it. Therefore, they propose a simple solution by complementing temporal difference with template matching. They claimed that although template matching also suffers from changes of the appearance of the target object, the combination of both methods eliminates all these handicaps [81].

For the domain of video tracking, Trucco and Plakas prepared a rich survey in which they discuss various methods of motion tracking in videos. They categorized some of the published tracking methods into the categories of window tracking, feature tracking, planar rigid shapes method, solid rigid object method, tracking deformable contours, and visual learning. In window tracking, the motivation is to find a region of correlation at time, $t+1$ for each pixel at time frame, t, and to repeat this for all frames

[139]. Feature tracking is similar to window tracking with the difference that instead of windows, image features are tracked from one time frame to the next. Those image features can be edges, corners, contours, etc. [137]. In planar rigid shapes, 2D shapes such as rectangles and circles are detected, then tracked over the sequence of frames. Solid rigid objects method extends the tracking of 2D shapes in planar rigid shapes method to 3D objects extracted from the images through finding 3D translational and rotational vectors. In tracking deformable contours, the challenge that is addressed is shapes changing through consecutive frames. In this method, the change in the shapes is represented as snakes [67, 136] or B-splines [9]. Finally, unlike the previous methods which use *a priori* models to extract shapes, objects, and targets, *visual learning* method approaches to the tracking problem through learning shapes and other complex dynamics from a training set composed of videos and images [92]. The survey finalizes with a table that compares example applications of these methods [138].

CHAPTER 3

ACTIVE INFERENCE FOR DYNAMIC BAYESIAN NETWORK FOR TISSUE ENGINEERING

In this chapter we illustrate an application of active inference for dynamic Bayesian network on a tissue engineering problem. Given the basics of tissue engineering as illustrated in Chapter 2, we first describe the problem in this domain on which we have focused along with our motivation. Next we present our approach and empirical evaluation. We finalize this chapter with a discussion.

3.1 Problem Description and Motivation

Tissue engineering researchers experiment with various settings, including the number of initial stem cells, the shape of scaffold, the distance between host blood vessels and stem cells, etc., in order to gain a better understanding of the tissue growth process. Ultimately, the goal is understanding the entire phenomenon and to detect the optimal settings for engineering a healthy tissue so that the lost tissue in humans can be healed through tissue engineering. Laboratory experiments take a long time however, often weeks if not months. Therefore, researchers often need to stop them early and make predictions about its future growth had the experiments not been stopped. To gain the best understanding of tissue growth process, researchers dissect tissue and analyze cellular matrix and blood vessel network, inspecting the depth and density of blood vessel coverage of the tissue. When a tissue is dissected, obviously, the experiment cannot resume where it stopped. Hence researchers need to determine the ideal time to stop an experiment and be able to make reliable predictions about its future formation had it not been stopped.

In this domain, the initial settings of an experiment is provided by the tissue

engineering researchers; hence, the random variables for the initial time, $t = 0$, are observed. Stopping an experiment at time t and dissecting the tissue is equivalent to observing the values of the random variables at time t. The researchers would like to predict the status of the experiment at a future time T. Hence, the active inference problem here is how early can we stop the experiment (i.e., what is the smallest value of t) so that the uncertainty of our prediction about time T is acceptable, i.e., less than a provided threshold?

We next introduce our DBN model of vascularization and then present the active inference problem more formally.

3.2 Modeling Approach and Problem Formulation

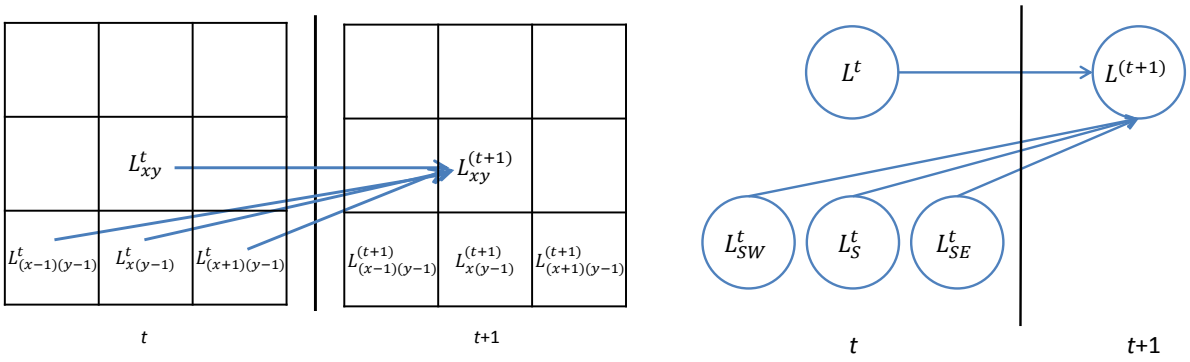
In this section, we first describe our DBN model for vascularization and then we discuss the active inference formulation for DBNs.

3.2.1 DBN Modeling of Vascularization in Tissue Growth. In a typical tissue engineering laboratory experiment, there are many factors considered: the shape and porosity of the scaffold that holds the tissue, the tissue cells that are in various cycles of their lives, and the distribution of the VEGF that is emitted by distressed cells, etc. In this study, we focus on the following question: given a group of distressed cells that signal their distress through VEGF, which we simply refer to as the *VEGF source*s (VS), and the closest host blood vessel, how likely are the VSs to be reached by new blood vessels that sprout from the host blood vessel?

In this study, we zoom in the region that is not vascularized yet. Hence, given one or more VSs and a host blood vessel, we model the vascularization using random variables that represent each location in a tissue grid as `Empty` (`E`), the tip of a new blood vessel `Tip Cell` (`TC`), or the body of the blood vessel `Stalk Cell` (`SC`). We elicit the DBN structure from domain experts and we experiment with numerous parameter

settings allowing the researchers to try various real and hypothetical situations. We next describe the structure and the parameters for the DBN.

3.2.1.1 The DBN Structure. In vascularization, at each step in time, the tip of a blood vessel elongates stochastically in the direction of the gradient of the VEGF, forming the body of the blood vessel. This corresponds to the following transition in our DBN: a location that is `TC` at time t becomes `SC` at time $t+1$ forming the body of the blood vessel, and whatever direction the blood vessel elongates towards becomes the new `TC`. Given a 2D grid of the space, following the assumption made in [88], we assume that the VSs are located at north whereas the host blood vessel is located in south, and hence the blood vessel elongates towards north-east, north, or north-west. Figure 3.1 illustrates the tissue grid, the dependencies between the random variables, and the corresponding 2-slice DBN.



(a) (b)

Figure 3.1. a) The tissue grid. The status of a location at time $t+1$ depends on the previous time t statuses of itself and its neighbors at its south-west, south, and south-east. b) The corresponding 2-slice DBN.

In this DBN, the status of a location at time $t+1$, L^{t+1}_{xy}, depends on the statuses of itself, L^t_{xy}, and the neighbors at its south-west, $L^t_{(x-1)(y-1)}$, south, $L^t_{x(y-1)}$, and south-east, $L^t_{(x+1)(y-1)}$, at previous time t. We simply refer to these neighbors generically as L^t_{SW}, L^t_S and L^t_{SE}. We next describe the parameter settings, i.e., the

conditional probability densities (CPDs), for this DBN.

3.2.1.2 The DBN Parameters. The typical vascularization process is as follows: a `Tip Cell` elongates stochastically in the direction of the VEGF source, VS, occupying an `Empty` location and forming the body of the blood vessel (`Stalk Cell`) during the process. Upon touched by VEGF, the body of the blood vessel might sprout a new blood vessel, i.e., a `Stalk Cell` might turn into a `Tip Cell`. Hence, if a location is

- a `Tip Cell` at time t, it elongates at time $t+1$ to an `Empty` location at one of its NW, N, or NE location, and leaves a `Stalk Cell` behind.
- a `Stalk Cell` at time t, it either remains a `Stalk Cell` or turns to a new `Tip Cell` to sprout a new blood vessel at time $t+1$.
- `Empty` at time t, whether it remains `Empty` or gets occupied by `Tip Cell` at time $t+1$ depends on whether there was a `Tip Cell` that can elongate to this location from at least one of its SW, S, and SE locations at time t.

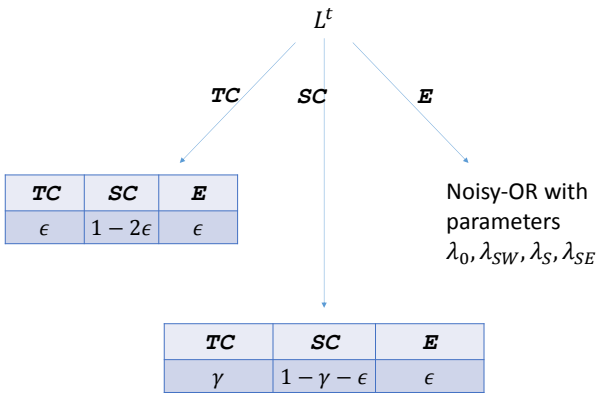


Figure 3.2. The tree CPD representation for $P(L^{(t+1)}|L^t, L^t_{SW}, L^t_S, L^t_{SE})$.

These transitions are best represented through a tree-CPD, which is shown in Figure 3.2. In this representation, we present the parameters in the order of $\langle \texttt{TC}, \texttt{SC}, \texttt{E} \rangle$.The CPD corresponding to the three above transitions is:

- $P(L^{(t+1)}|L^t = \texttt{TC}, L^t_{SW}, L^t_S, L^t_{SE}) = P(L^{(t+1)}|L^t = \texttt{TC}) = \langle \epsilon, 1-2\epsilon, \epsilon \rangle$
- $P(L^{(t+1)}|L^t = \texttt{SC}, L^t_{SW}, L^t_S, L^t_{SE}) = P(L^{(t+1)}|L^t = \texttt{SC}) = \langle \gamma, 1-\gamma-\epsilon, \epsilon \rangle$
- $P(L^{(t+1)}|L^t = \texttt{E}, L^t_{SW}, L^t_S, L^t_{SE}) = \text{noisy-OR}(L^t_{SW}, L^t_S, L^t_{SE})$

where ϵ is a small noise parameter and γ is the new sprout probability. Whether an E location at time t gets occupied with the tip of a blood vessel (TC) at time $t+1$ is modeled as a noisy-OR of its parents located at its SW, S, and SE neighbors: $P(L^{(t+1)} = \texttt{TC}|L^t = \texttt{E}, L^t_{SW}, L^t_S, L^t_{SE})$ is a noisy-OR of L^t_{SW}, L^t_S, L^t_{SE}, with parameters λ_0, λ_{SW}, λ_S and λ_{SE}. λ_0 is a leak parameter, and λ_{SW}, λ_S and λ_{SE} represent the possibility that a TC in the SW, S, or SE will elongate to this location.

The magnitude of λ_{SW}, λ_S and λ_{SE} are determined by i) how far the VEGF can travel before it completely dissipates and ii) the magnitude of the VEGF gradient. If the VEGF cannot reach a TC, then TC follows a path uniformly at random, exploring its surroundings. If TC is reached by VEGF, then the closer the TC to the source of VEGF , VS, the higher the VEGF gradient, and hence the λ values become more skewed towards the VS. The further away from the VS, the more uniform the λ values get.

More formally, let d_{xy} be the distance of location L_{xy} to VS and let d_V represent the maximum distance that the VEGF can travel before it dissipates (or becomes negligible). If $d_{xy} > d_V$, then because VEGF cannot reach L_{xy}, the tip cell at L_{xy} has no clue as to whether it should grow towards NW, N, or NE, and hence it has equal probability in either direction: $\lambda_{(x-1)(y+1)} = \lambda_{(x)(y+1)} = \lambda_{(x+1)(y+1)} = \frac{1}{3}$. If $d_{xy} \leq d_V$,

then the λ values become skewed towards VS. Without loss of generality, assume that the VS is located at NE with respect to xy. Then $\lambda_{(x-1)(y+1)} = \lambda_{(x)(y+1)} = \frac{d_{xy}}{3d_V}, \lambda_{(x+1)(y+1)} = \frac{d_{xy}}{3d_V} + (1 - \frac{d_{xy}}{d_V})$, and hence the `TC` has a higher probability moving NE.

It is important to note that the λ formalism discussed above is only an approximation to the reality. The VEGF gradient is one of many factors that effect the direction of the blood vessel growth. Hence, the probabilistic framework allows researchers to "sweep the unaccounted and uncontrolled factors under the probability rug." Additionally, how far the VEGF travels depends on a number of factors including the scaffold porosity. Therefore, we do not set d_V parameter to a fixed value; rather, we allow the researchers to experiment with various hypothetical values to see how it effects vascularization.

In collaboration with the tissue engineering researchers, we developed the first prototype of our DBN model of tissue development in our workshop paper [71]. In our first prototype, we made a number of simplifying assumptions. For example, we assumed that the λ values were fixed across the grid. In our model of the second phase, we allow a distressed cell that can emit VEGF, which in turn results in non-uniform λ values across the grid. Another limitation existing in the first model was the grid size. In that work, largest size applicable in the proposed model was 9x9 whereas in the extended version we reach sizes beyond this limit such as 51x51. Additionally, we formulate and experiment with active inference in the second phase. As it is described in the next section we propose a novel approach to measure uncertainty in temporal graphical models. Furthermore, using this uncertainty measure we formulate a trade-off on two conflicting objectives. Using uncertainty measure and formulated trade-off we applied some common search methods to detect optimal time for decision making

as this method called active inference.

3.2.2 Active Inference for DBNs.

3.2.2.1 Motivation and Objective Function. In this section, we formalize the question of "*given an uncertainty threshold of σ, how early can a tissue engineering experiment be stopped so that the prediction uncertainty over the tissue grid for a target time slice T is below σ*?" More formally, we need to find t^* such that:

$$t^* = \underset{t<T}{\operatorname{argmin}}\, UNC\left(P\left(\mathcal{L}^T|l^0, l^t\right)\right) < \sigma$$

where $UNC\left(P\left(\mathcal{L}^T|l^0, l^t\right)\right)$ is the prediction uncertainty over the target time slice T, σ is the maximum uncertainty acceptable to the researcher, $\mathcal{L}^T$ is the collection of all locations at T, l^t is the observation (i.e., the values of all the random variables L^t_{xy} at time t) we would get once we stop the lab experiment and dissect the tissue. Of course, we do not know what the status of the experiment would be at time t unless we actually stop the experiment. Therefore, a standard technique is to take an expectation over all possible outcomes l^t:

$$t^* = \underset{t<T}{\operatorname{argmin}} \sum_{l^t} P\left(\mathcal{L}^t = l^t|l^0\right) UNC\left(P\left(\mathcal{L}^T|l^0, l^t\right)\right) < \sigma$$

A challenge here is, however, that if the number of all locations at time t is m, then l^t ranges over 3^m possible assignments (each location can be `TC`, `SC`, or `E`), which makes taking the expectation clearly intractable. There are two viable approximations that are common in the literature. One is to sample potential assignment values using the probability distribution $P\left(\mathcal{L}^t|l^0\right)$ and take the mean of $UNC\left(P\left(\mathcal{L}^T|l^0, l^t\right)\right)$. A second approach is to use l^t where $P\left(\mathcal{L}^t|l^0\right)$ is maximum (i.e., the MAP assignment). Note that finding l^t where $P\left(\mathcal{L}^t|l^0\right)$ is maximum does not necessarily require us to enumerate all possible l^t values, when i) each location at time t is independent of other locations at time t given their parents, and ii) there are no future, $t' > t$, observations.

Both assumptions hold for our DBN structure.

3.2.2.2 Uncertainty Definition. There are a number of possible approaches for defining uncertainty over the target time slice, $UNC\left(P\left(\mathcal{L}^T|l^0,l^t\right)\right)$. One typical approach is to use entropy. However, entropy cannot capture correctly the cases that are misclassified with high certainty. In our domain, the researchers are interested in predicting the path that a new sprout from the host blood vessel follows and hence we formulate uncertainty as the conditional error of the most probable blood vessel path that originate from the host blood vessel.

More formally, let $p = \langle p_1, p_2, ..., p_{|path|}\rangle$ be a path of length $|path|$ that represents a connected sequence of locations representing a blood vessel, whose tip $p_{|path|}$ is a `TC` and whose body $\langle p_1, p_2, ..., p_{|path-1|}\rangle$ are `SC`s. Then, the conditional error over this path p being a blood vessel is:

$$UNC(p|l^t,l^0) = \left(\sum_{i=1}^{|path|-1}\left(1 - P(p_i = \texttt{SC}|l^t,l^0)\right)\right) + \left(1 - P(p_{|path|} = \texttt{TC}|l^t,l^0)\right)$$

The uncertainty over the target time locations $\mathcal{L}^T$ is then defined as the conditional error of the most probable blood vessel path at time T:

$$UNC\left(P\left(\mathcal{L}^T|l^0,l^t\right)\right) = \underset{p\in\mathcal{L}^T}{\operatorname{argmin}}\, UNC(p|l^t,l^0) \tag{3.1}$$

3.2.2.3 Search. To find the optimal t^* to stop a laboratory experiment so that uncertainty for target time T is below threshold σ, we need to search for the $0 < t < T$ for which $UNC\left(P\left(\mathcal{L}^T|l^0,l^t\right)\right) < \sigma < UNC\left(P\left(\mathcal{L}^T|l^0,l^{t-1}\right)\right)$. However, such a search can be computationally expensive. For each t value we try, we need to:

- *Step 1:* Find the most-likely assignment l^t for $\mathcal{L}^t$, i.e., find $\underset{l^t}{\operatorname{argmax}}\, P(\mathcal{L}^t = l^t|l^0)$.

Given the Bayesian network structure, this requires us to run inference from $t' = 0$ to $t' = t$; any variables for $t' > t$ are irrelevant. However, given two times $t_1 < t_2$, the computations done for $\underset{l^{t_1}}{\text{argmax}}\, P(\mathcal{L}^{t_1} = l^{t_1}|l^0)$ can be reused for computing $\underset{l^{t_2}}{\text{argmax}}\, P(\mathcal{L}^{t_2} = l^{t_2}|l^0)$. For example, if a sampling is used for inference, the counts can be stored and reused for all $0 \leq t \leq T$. Therefore, this step can be done once for the whole search process.

- *Step 2:* Compute $UNC\left(P\left(\mathcal{L}^T|l^0, l^t\right)\right)$, which requires us to run inference with l^t as observed, find the most-probable blood vessel path of length T at time T, and compute conditional error for this blood vessel path. Unfortunately, for this phase, computations needed for $P\left(\mathcal{L}^T|l^0, l^t\right)$ cannot be shared between different t values, and hence, this step needs to be repeated for every t value the search algorithm tries.

In this active inference setting, the observations are always for past time slices; i.e., we need to compute $P\left(\mathcal{L}^t|l^0\right)$ where $t > 0$ and $P\left(\mathcal{L}^T|l^0, l^t\right)$ where $t < T$. We assume that once the laboratory experiment is stopped at time t and the tissue is dissected, all locations at time t can be observed by the experimenter. Hence, forward sampling can be conveniently utilized for both probability computations. Then, the inference cost for computing $P(\mathcal{L}^{t'}|l^t)$ is

$$IC(t'|t) = m \times (t' - t) \times s$$

where m is the number of locations on each time slice and s is the number of samples per location. The inference cost for *Step 1* is independent of t and it is $IC(T|0)$ because sampling is done once for all time slices and the computations are shared between different time slices. The inference cost for *Step 2* depends on t and it is $IC(T|t)$. Hence, a search algorithm that tries multiple time steps will incur the first cost once and it will incur the second cost for each t tried. More formally, the

aggregated inference cost for a search algorithm to find the optimal t^* is:

$$ASC(t^*|search) = IC(T|t=0) + \sum_{t \in search} IC(T|t) \tag{3.2}$$

Assuming the uncertainty over the target time slice decreases as the evidence is gathered later in time (i.e., the later the experiment is stopped, the more reliable the predictions about the target time slice get), viable search algorithms are:

- Forward search (`ForS`), starting with $t = 1$ and ending when the uncertainty goes below σ,
- Backward search (`BackS`), starting with $t = T - 1$ and ending when the uncertainty goes above σ,
- Binary search (`BinS`), starting with end points, and halving the space at each iteration, and
- Line search (`LineS`), iteratively fitting a line to the uncertainties of the latest known (computed) time points and trying to pinpoint the optimal time t^* accordingly. Line search iteratively fits a line to the two known/computed end points of the interval that contain t^*.

We compare these search algorithms in terms of their aggregated inference costs (Equation 3.2) both empirically and analytically in the rest of the article.

3.3 Empirical Evaluation

In this section, we present inference results of our DBN model on various real and hypothetical settings that are obtained by varying the location of the VEGF source, VS, the number and locations of initial sprouts as `Tip Cell`s (`TC`s), and the maximum distance the VEGF can travel, d_V.

3.3.1 Experimental Setup. Given an initial setup specified by the experimenter,

which is specified as the values of all the random variables at time $t = 0$, i.e., the locations that are TC, SC, and E at $t = 0$, given the initial location of the distressed cell VS, and the maximum distance that the VEGF can travel, d_V, we compute the probability distribution of $\langle \texttt{TC}, \texttt{SC}, \texttt{E} \rangle$ for each location for each time slice $0 \leq t \leq T$. We present detailed results for 5×5 and 9×9 grids for illustrative purposes first. Then we present active inference results using a much larger 51×51 grid, which is more realistic. Note that these grids are relatively small compared to the full tissue space, because each grid does not represent the full tissue space but rather represents a zoomed region of a single VS, because the task is to figure out the likelihood of this single VS being reached by a blood vessel.

The grid is drawn in such a way that the VS is at the top/north of the grid ($y = Y - 1$) and the blood vessel sprout(s) (TC) are located at the bottom/south of the grid ($y = 0$). We experimented with various conditions where the VS is located in the middle versus in the corners, the TC is in the middle or the corners, and there is only one or more TCs in the grid. We also varied how far the VEGF can travel (d_V). We chose three representative cases where the VEGF can travel short, medium, and long (in comparison to the grid height) distances $d_V = Y/4$, $d_V = Y$, and $d_V = 4Y$.

3.3.2 Prediction Results. Figure 3.3 shows the probability of each location being occupied by a blood vessel cell (i.e., $P(L^t = \texttt{TC}|l^0) + P(L^t = \texttt{SC}|l^0)$) for $t = 0, 1, 2, 3, 4$ for a 5×5 grid. The case where VEGF can travel only a short distance, $d_V = Y/4$, is shown at the top and the case where VEGF can travel a long distance, $d_V = 4Y$, is shown at the bottom. For both cases one VS is located at the middle north location and one TC is located in the middle south.

Note that λ parameters denote the likelihood of TC elongating in one of NW, N, or NE directions and the values of these parameters depend whether the TC is reached by VEGF and if so, how close the TC is to the VS, the source of the VEGF.

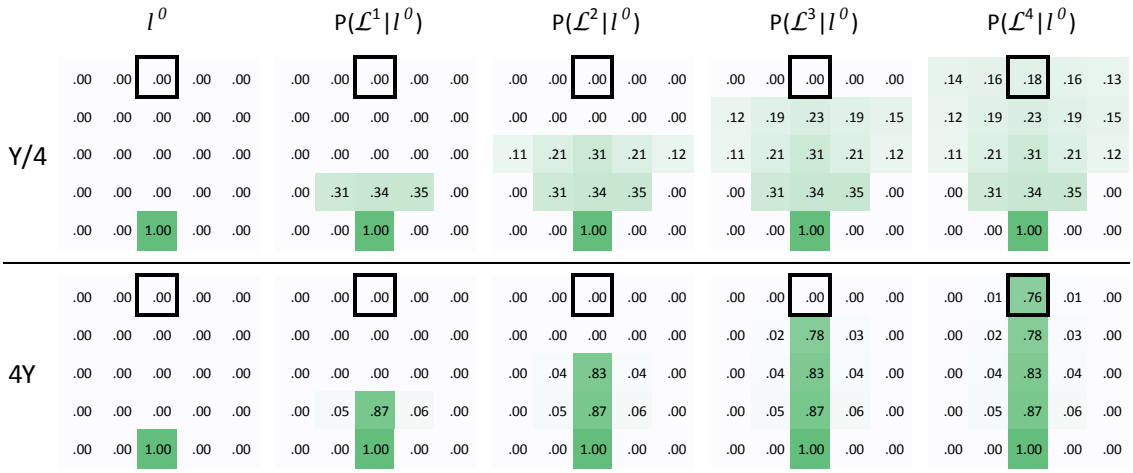


Figure 3.3. Cell probabilities (the sum of TC and SC) for the 5×5 grid at all time slices for $d_V = Y/4$ and $d_V = 4Y$.

For $d_V = Y/4$, any location that is further away from the VS as much as $Y/4$ or more has uniform λ values and hence the blood vessel has equal likelihood of growing in all three directions, resulting in uniform probabilities (the top of Figure 3.3). For $d_V = 4Y$, all locations are in the VEGF diffusion range, causing the λ values to be more skewed in the gradient of the VEGF. In this case, we see that the TC simply follows the gradient to reach the distressed cell. For $d_V = Y/4$, the probability that the TC reaches the distressed cell is only 0.18 whereas for $d_V = 4Y$, the probability is much higher: 0.76.

We next present results of different placings of VS and TC on a 9×9 grid (we present only the $d_V = 4Y$ case as the $d_V = Y/4$ case results simply in uniform probabilities). The top of Figure 3.4 shows a setting where the VS is on the north-west corner and a single TC is on the south-east corner. The bottom of the same figure shows the results for the case of one VS in the middle north, one TC in the south-east corner and another TC in the south-west corner.

Comparing to the 5×5, the 9×9 grid is larger in both dimensions and furthermore, on top the top figure, the VS and TC are placed on the opposite ends

of a diagonal. This setting doubles the distance between VS and TC comparing with the previous case and hence the likelihood of VS being reached by a blood vessel is much smaller, 0.37 versus 0.76. At the bottom, however, the chance of VS being reached by a blood vessel is much higher then the case on top since there are two initial TCs and VS is not as further away. These results illustrate the obvious: that as the VEGF travels longer distances, the chance of a blood vessel finding the VS increases. When multiple initial sprouts are involved, the chance rises even more. What is more, however, is that it associates a probability with these scenarious that can be acted upon by the tissue engineering researchers.

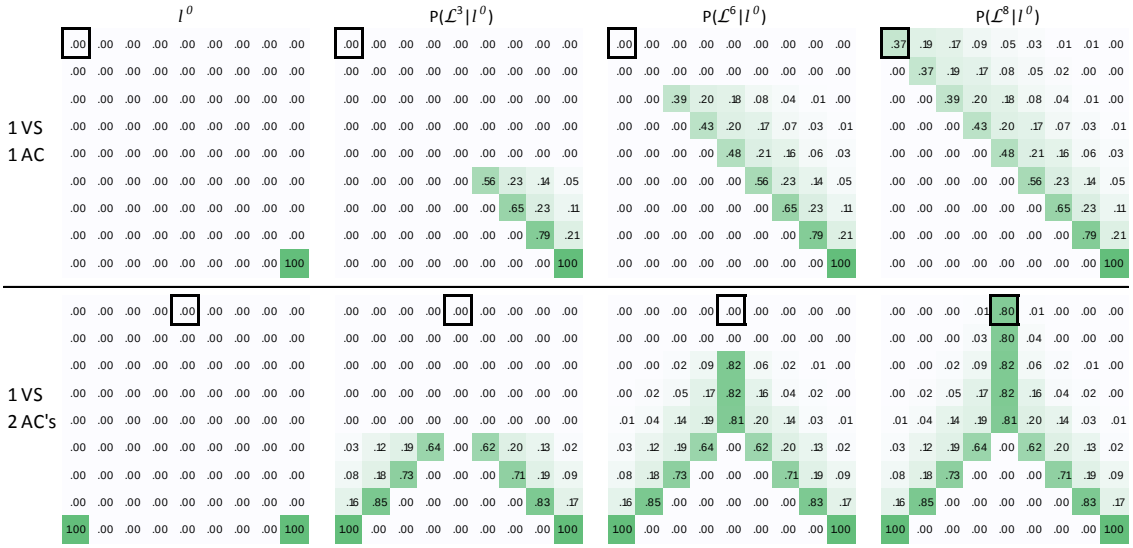


Figure 3.4. Cell probabilities for the 9×9 grid at 4 selected time slices for maximum travel distance $= 4Y$.

3.3.3 Discussion. Evaluating mathematical models of tissue engineering is fairly challenging for several reasons: there are many uncontrolled and unaccounted factors that mathematical models simply do not take into account. Therefore, the process looks just too stochastic from the modeling perspective, and comparison by a set of controlled experiments is not straightforward. Therefore, rather than evaluating precision of locations of the blood vessels, general characteristics are evaluated, such as how much branching occurs, the mean and the maximum blood vessel invasion

depth, blood vessel length density, etc.

3.3.3.1 Laboratory Experiments. We next present laboratory experimental results where we focus on the invasion depth, i.e., the depth of vascularized tissue growth into the environment. An experimental study was previously performed using porous polymer scaffolds containing gradients of growth factors *in vivo* [2]. More detail on the experimental conditions can be found in [2]. Briefly, gradient scaffolds with varying growth factor doses (0, 2, 20 and 200 ng) of PDGF-BB (a growth factor, of which VEGF is a subfamily) were investigated on a rodent model and vascularization was evaluated at 1, 3, and 6 weeks post implantation. Harvested samples were stained, i.e., the tissue was colored with dyes (in this case with hematoxylin and eosin (H&E)) to enhance visibility and contrast in the microscopic imaging. H&E stain cell nuclei and the tissue structure purple and pink, respectively.

Stained samples were imaged to quantify tissue invasion depth using an Axiovert 200 inverted microscope (1.10 µm/pixel). The depth of tissue invasion was measured as the straight-line distance from the underlying host tissue to the deepest location where tissue could be seen within the scaffolds. For these experiments, five animals were sacrificed at each time point (1 week, 3 weeks, and 6 weeks). Each animal received four implants each one corresponding to a different growth factor concentration (0, 2, 20, and 200ng). This procedure resulted in five samples per growth factor concentration per time point. We took measurements from three different locations (right, middle, and left) within each sample. Therefore, we have $5 \times 3 = 15$ measurements for each condition (dose & time). In each measurement, we calculated tissue invasion depth statistics for various threshold ranges (Table 3.1). Statistics were calculated by dividing the number of times tissue invasion reaches an identified threshold to the total number of measurements of the condition.

Blood vessels within the scaffold were labeled using a fluorescent dye (Alexa

Table 3.1. Statistics of tissue invasion depth.

0 ng	**<1000 µm**	**1000 µm**	**1250 µm**	**1500 µm**	**>1750 µm**
Week 1	1	0.2	0.13	0.07	0
Week 3	1	0.53	0.13	0	0
Week 6	1	0.73	0	0	0
2 ng	**<1000 µm**	**1000 µm**	**1250 µm**	**1500 µm**	**>1750 µm**
Week 1	1	0.53	0.33	0.2	0.07
Week 3	1	0.8	0.73	0.53	0.27
Week 6	1	0.67	0.27	0.07	0
20 ng	**<1000 µm**	**1000 µm**	**1250 µm**	**1500 µm**	**>1750 µm**
Week 1	1	0.73	0.4	0	0
Week 3	1	0.93	0.67	0.4	0.2
Week 6	1	0.87	0.73	0.27	0.2
200 ng	**<1000 µm**	**1000 µm**	**1250 µm**	**1500 µm**	**>1750 µm**
Week 1	1	0.93	0.8	0.47	0.2
Week 3	1	1	0.87	0.67	0.6
Week 6	1	1	1	0.4	0.33

Fluor 647-isolectin) and imaged using confocal microscopy. Representative images of blood vessels within gradient scaffolds (20 ng PDGF-BB) for weeks 1, 3 and 6 are shown in Figure 3.5.

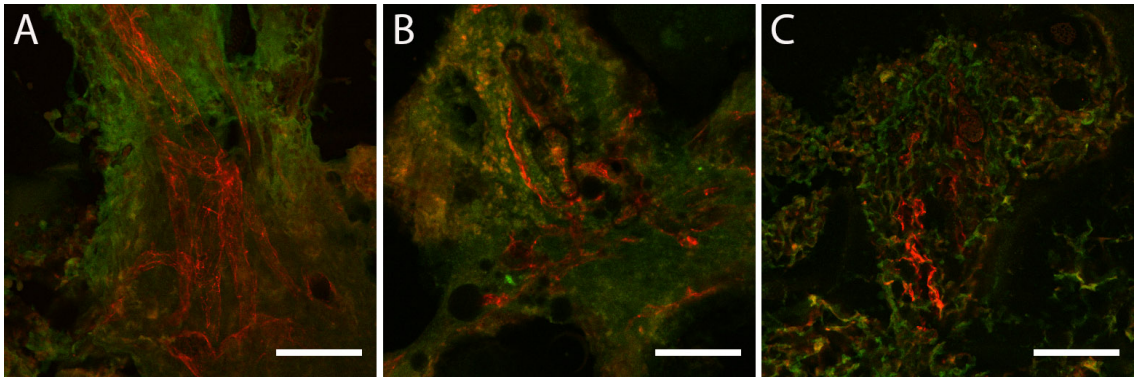


Figure 3.5. Blood vessel invasion within gradient scaffolds at (A) week 1, (B) week 3, and (C) week 6. All images are for the 20 ng PDGF-BB case. Red shows isolectin labeled blood vessels and green is autofluorescence of the tissue. The large black areas are the scaffold structure. Scale bars are 100 µm.

As these results in Table 3.1 illustrate, the higher the growth factor dose (the lower rows), the higher the likelihood of deeper invasion of the tissue (the columns on the right). For example, at week 6, the ratios of samples where an invasion depth of 1250 µm was reached are 0%, 27%, 73%, and 100% corresponding to 0 ng, 2 ng, 20 ng, and 200 ng doses of growth factor. Note that in some cases, the invasion depth decreases between 3 and 6 weeks due to regression and pruning, i.e., cell death.

We cannot directly compare these numbers (Table 3.1) with the numbers in Figure 3.3, because our model focuses on specific regions whereas these experimental results show aggregate information. However, these data support the claim that our DBN model captures the phenomenon that the higher the growth factor concentration is (Figure 3.3 bottom figure), the higher the likelihood of reaching distressed cells.

3.3.3.2 Agent-Based Models. A direct comparison of our DBN model with the laboratory experiments is challenging, as was explained above. However, there are other mathematical models of tissue engineering applications where one-to-one evaluation is more straightforward. One such mathematical modeling approach is agent-based modeling. For example, [5] developed an agent-based model to investigate blood vessel formation in biomaterial scaffolds, and the model was improved progressively through constant revision and incorporating new experimental findings [5, 87, 88, 130].

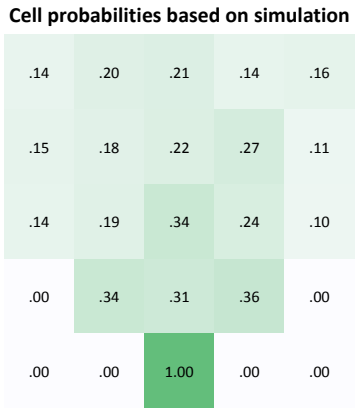
The agent-based models have many advantages over dynamic Bayesian networks. Different cell types such as stem cells, tissue cells, and the cells forming blood vessels can be represented through different agents with specific rules governing their behavior. This allows the modeler to impart domain knowledge easily into the system. For example, a cell going through its life cycle can exist in different states and perform actions such as migration, proliferation, or differentiation. The cell state such as being hypoxic or not can be determined by the presence (or lack thereof) of

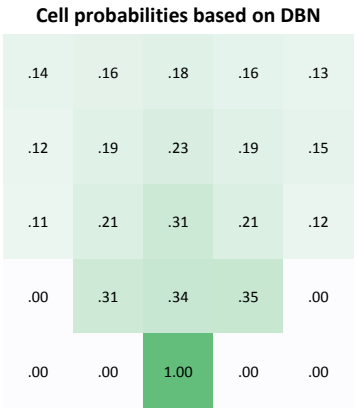
nearby blood vessels that are also represented by agents that have their own rule set.

A big challenge in using agent-based models is the need for many simulations to observe the average behavior of the system. In order to observe the average behavior, the researcher repeats the same simulation multiple times, starting with the same initial conditions but varying the random seed. The number of trials is often decided on an ad-hoc manner. Graphical models have the obvious advantage over such simulations in observing average behavior because that is what the probabilistic graphical models are obviously designed for. Further, and more importantly, graphical models allow for "what if" analysis more effectively by enabling one to provide hypothetical evidence values for future time slices and then one can reason both backward and forward in time. Starting an agent-based model in future and running it backwards, however, is almost impossible, unless one designs a complete set of new rules that takes agents back in time.

We next provide a comparison of DBN inference results to average behavior of the agent based model of Artel et al. [5]. The average behavior of the agent-based model is obtained by running multiple simulations (200 in this case) and averaging the observed results. In Figure 3.6, on the left, we present cell probabilities at 5^{th} time tick at each location after 200 runs of the agent-based model simulation and in the middle we present cell probabilities at 5^{th} time slice of the DBN model. These results show that both models result in the same average behavior. We computed Jensen-Shannon divergence between the results of these two models and present it as a table on the right of Figure 3.6. As these results show, the differences between the two models are very small, and the differences stem from the approximation errors of the average behavior from multiple simulations.

In addition to matching the results of the agent-based model, the DBN allows for more complicated queries that the agent-based models cannot readily handle. For





Jensen-Shannon Divergence

3.80E-05	1.52E-03	1.03E-03	5.66E-04	9.23E-04
1.39E-03	1.20E-04	2.35E-04	5.79E-03	3.30E-03
1.49E-03	7.12E-04	5.16E-04	9.31E-04	7.38E-04
0	5.16E-04	7.40E-04	1.98E-05	0
0	0	0	0	0

Figure 3.6. The comparison of cell probabilities obtained from 200 runs of simulation with our DBN model.

example, we can observe the behavior of the underlying vascularization laboratory experiment at time t and enter the observations as evidence into the DBN model. Further, this ability to enter the state of the experiment as evidence into DBN allows us to perform active inference, which we discuss next in the following section.

3.3.4 Active Inference Results. In this section, we provide empirical results for investigating the question of "*given an acceptable uncertainty threshold of σ, how early can a tissue engineering experiment be stopped so that the prediction uncertainty for a target time T is below σ*?" We first present results exploring how uncertainty, as defined in Equation 3.1, is affected by d_V and how it varies by t. Then, we compare the computation cost of various search algorithms that search for the optimal time to stop an experiment.

3.3.4.1 The Uncertainty Distribution. In this experiment, we computed uncertainty as defined in Equation 3.1 on a 51×51 grid. We placed one VS in the middle north and one `TC` in the middle south. We experimented with $d_V = Y/4$, $d_V = Y$, and $d_V = 4Y$ to observe the relation between d_V and uncertainty. Figure 3.7 shows the uncertainty in the y axis, assuming the lab experiment is stopped at time t (the x axis).

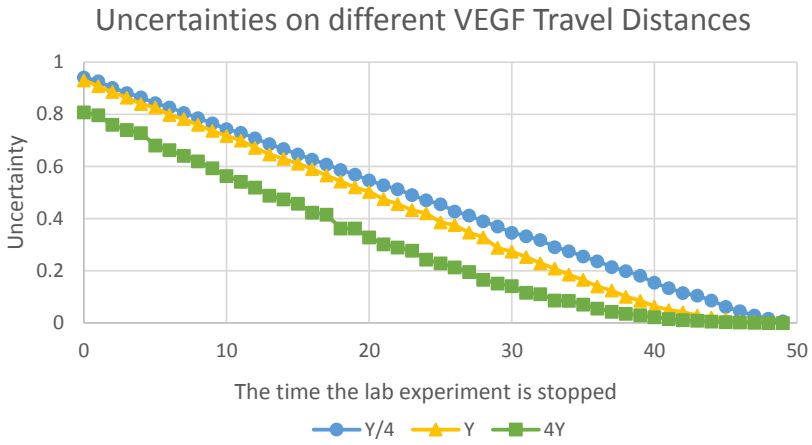


Figure 3.7. Uncertainty at time slice T for each time slice candidate of observation, using various d_V values.

As expected, as we stop the tissue engineering experiment at later time slices, the uncertainty for the target time goes down. Additionally, the longer the VEGF can travel, the more skewed the λ values become and hence the blood vessel path becomes more predictable, i.e., less uncertain.

Next, we tackle the question of finding the optimal t^*, the earliest time t that guarantees a prediction uncertainty below the threshold of σ. This question can be easily answered if we compute the expected uncertainty for each candidate time t, like we did in Figure 3.7. However, computing uncertainty for a candidate time t is computationally expensive, as was discussed and formulated in the approach section. Next, we compare various search algorithms on how they fare in minimizing this computational cost.

3.3.4.2 Uncertainty Search. We compared the search methods we described in the approach section: forward search (`ForS`), backward search (`BackS`), binary search (`BinS`), and line search (`LineS`) in terms of the amount of computation (Equation 3.2) they would require to find t^*. We tested how the search algorithms would compare if the ideal time t^* was $t = 1$, $t = 2$, ..., $t = T - 1$. We experimented with both

$d_V = Y/4$ and $d_V = 4Y$ cases, but we show only the $d_V = Y/4$ case, as the other case is similar.

Figure 3.8 shows the aggregated computation cost of each search method. The x axis represents the hypothetical ideal time to stop an experiment and the y axis shows the aggregated search cost a search algorithm would incur to find that ideal time. As expected, `ForS` incurs more cost and `BackS` incurs less cost as we move t^* to later time slices, and `BinS` outperforms both `ForS` and `BackS`. `LineS` performs the best because it fits a function to the so-far-computed uncertainty values and tries to pinpoint t^* using this function. We also see that `LineS` outperforms `BinS` for each t^*.

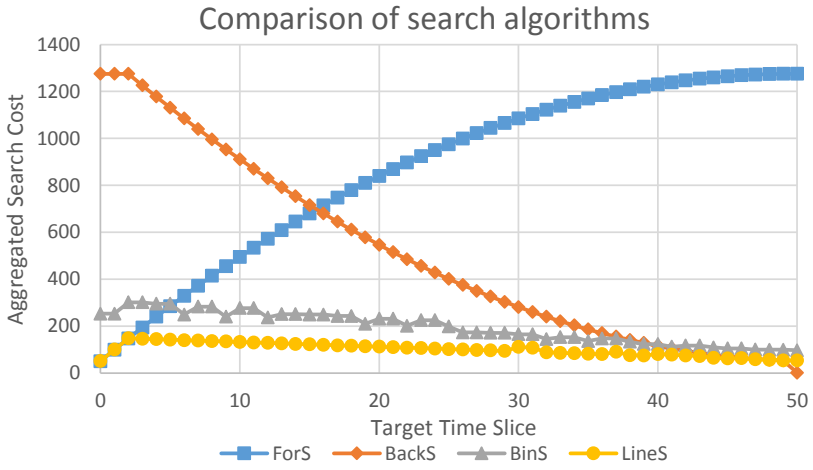


Figure 3.8. Aggregated computation cost for each search method when $d_V = Y/4$.

3.4 Analytical Evaluation

In this section, we provide closed-form solutions to the aggregated search costs (Equation 3.2) for the search algorithms `ForS`, `BackS`, `BinS`, and `LineS`. We make the reasonable assumption that uncertainty decreases as we provide evidence for later time slices. That is, $UNC\left(P\left(\mathcal{L}^T|l^0, l^{t+1}\right)\right) < UNC\left(P\left(\mathcal{L}^T|l^0, l^t\right)\right)$ for $0 < t < T$. This assumption is verified to hold for our domain as was shown in Figure 3.8.

In the active inference problem, we are searching for t^* where $t^* = \text{argmin}_{t<T}$ $UNC\left(P\left(\mathcal{L}^T|l^0, l^t\right)\right) < \sigma$. `ForS` tries $t = 1, 2, \cdots, T-1$ in increasing order till it finds the first t that satisfies the uncertainty requirement. Therefore, `ForS` will try $t = 1, 2, \cdots, t^*$. `BackS` tries $t = T-1, T-2, \cdots, 1$ in decreasing order till it finds the first t that violates the uncertainty requirement. That is, `BackS` will try $t = T-1, T-2, \cdots, t^*, t^*-1$. `BinS` will perform search by halving the search space at each iteration. That is, it will first try $t = T/2$ and depending on the uncertainty at this time, it will try either $t = T/4$ or $t = 3T/4$, and so on, till it tries t^* and either one of t^*-1 or t^*+1. `LineS` tries $t = T-1$ first and then fits a line to the uncertainty at time $t = 0$, which is given, and tries to pinpoint the exact uncertainty threshold. If the pinpoint uncertainty is larger than the threshold, it fits a new line between the pinpoint and $t = T-1$, otherwise between the pinpoint and $t = 0$. It iterates narrowing the search interval down until σ is clamped between two latest pinpoints.

3.4.1 Forward Search. The aggregated search cost for `ForS`, $ASC(t^*|\texttt{ForS})$ is:

$$\begin{aligned} ASC(t^*|\texttt{ForS}) &= IC(T|t=0) + IC(T|t=1) + ... + IC(T|t=t^*) \\ &= m \times s \times [(T-0) + (T-1) + ... + (T-t^*)] \\ &= m \times s \times [(t^*+1) \times T - \frac{t^*(t^*+1)}{2}] \\ &= m \times s \times (t^*+1)(T - \frac{t^*}{2}) \end{aligned}$$

Note that $0 < t^* < T$ and hence each of the terms $(T-0), (T-1), \ldots, (T-t^*)$ in the second line of this equation are positive and the number of terms grows in t^*. Therefore, the ASC for `ForS` grows as the ideal time to stop the experiment is closer to the target time T as expected and as has also been empirically validated

and shown in Figure 3.8.

3.4.2 Backward Search. Similarly, the aggregated search cost of `BackS` is:

$$
\begin{aligned}
ASC(t^*|\texttt{BackS}) &= IC(T|t=0) + IC(T|T-1) + IC(T|T-2) + \cdots + \\
&\quad IC(T|t^*) + IC(T|t^*-1) \\
&= m \times s \times [(T-0) + (T-T+1) + (T-T+2) + \cdots + \\
&\quad (T-t^*) + (T-(t^*-1))] \\
&= m \times s \times \big[T + \big(T-(T-1)\big) + \big(T-(T-2)\big) + \cdots + \\
&\quad \big(T-(t^*-1)\big)\big] \\
&= m \times s \times \big[T + T \times \big((T-1)-(t^*-1)+1\big) - [(T-1)+ \\
&\quad (T-2) + \cdots + (t^*-1)]\,\big] \\
&= m \times s \times \left[T + T[T-t^*+1] - \frac{(T-t^*+1)(T+t^*-2)}{2}\right] \\
&= m \times s \times \left[T + \frac{(T-t^*+1)(T-t^*+2)}{2}\right]
\end{aligned}
$$

Following similar reasoning, each of terms $(T-0), (T-T+1), (T-T+2), \ldots, (T-(t^*-1))$ in the second line are positive and the number of terms decreases in t^* resulting lower cost as the ideal time to stop the experiment gets closer to the target time T.

The ASC for `ForS` is monotonously increasing in t^* whereas it is monotonously decreasing for `BackS`. Next, we determine t^* for which `ForS` and `BackS` have equal cost.

$$(t^* + 1)\left(T - \frac{t^*}{2}\right) = T + \frac{(T - t^* + 1) \times (T - t^* + 2)}{2}$$
$$-\frac{1}{2}t^{*2} + (T - \frac{1}{2})t^* + T = T + \frac{(T - t^* + 1) \times (T - t^* + 2)}{2}$$
$$-\frac{1}{2}t^{*2} + (T - \frac{1}{2})t^* + T = T + \frac{t^{*2} - (2T - 3)t^* + T^2 + 5T + 2}{2}$$
$$2t^{*2} + (-4T - 2)t^* + T^2 - 3T + 2 = 0$$

The solution of this quadratic equation is:

$$t^* = \frac{4T + 2 \pm \sqrt{(-4T - 2)^2 - 4 \times 2 \times (T^2 - 3T + 2)}}{4}$$
$$= T + \frac{1}{2} \pm \sqrt{\frac{T^2}{2} - \frac{T}{2} + \frac{5}{4}}$$

The feasible solution for which $t^* < T$ is:

$$t^* = T + \frac{1}{2} - \sqrt{\frac{T^2}{2} - \frac{T}{2} + \frac{5}{4}}$$

For large T values, the t^* for which `ForS` and `BackS` have equal aggregated inference cost is equal to:

$$\begin{aligned} t^* &\approx T - \frac{T}{\sqrt{2}} \\ &= T(1 - 1/\sqrt{2}) \\ &\approx 0.3T \end{aligned} \tag{3.3}$$

That is, for $t^* < 0.3T$, `ForS` is more efficient than `BackS`, and for $t^* > 0.3T$, it is better to use `BackS`. For $T = 50$, `ForS` and `BackS` are equal on $t^* = 15$ which is roughly the value we see in Figure 3.8.

3.4.3 Binary Search. For binary search, because the actual steps depend on

where t^* lies, we provide closed-form solutions for i) the worst-case, ii) the best-case, and iii) average-case scenarios.

3.4.3.1 Worst-Case Scenario. Because the inference cost of earlier steps is larger than the cost of the later time steps, in the worst case, the binary search keeps trying earlier time steps $t = \frac{T}{2}, \frac{T}{4}, \cdots, \frac{T}{2^{\lceil \log_2 T \rceil}}$. Let $n = \lceil \log_2 T \rceil$. The inference cost is then:

$$\begin{aligned} ASC(t^* = 0|\texttt{BinS}) &= m \times s \times \left[T + \left(T - \frac{T}{2} \right) + \left(T - \frac{T}{4} \right) + \cdots + \left(T - \frac{T}{2^n} \right) \right] \\ &= m \times s \times \left[T + nT - T \left(\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^n} \right) \right] \\ &= m \times s \times \left[T + nT - T \left(\frac{1}{2} + \frac{1}{4} + ... + \frac{1}{2^n} \right) \right] \\ &= m \times s \times \left[T + nT - T \left(\frac{2^n - 1}{2^n} \right) \right] \end{aligned}$$

$\frac{2^n - 1}{2^n} \approx 1$ for large n. So the equation becomes:

$$\begin{aligned} ASC(t^* = 0|\texttt{BinS}) &\approx m \times s \times [T + nT - T] \\ &= m \times s \times nT \\ &= m \times s \times \lceil \log_2 T \rceil T \end{aligned}$$

3.4.4 Best Case. Because the inference cost of later time steps is smaller than the cost of the earlier time steps, in the best case, the binary search keeps trying later time steps $t = \frac{T}{2}, \frac{3T}{4}, \cdots, \frac{(2^{\lceil \log_2 T \rceil} - 1)T}{2^{\lceil \log_2 T \rceil}}$. Let $n = \lceil \log_2 T \rceil$. The inference cost is then:

$$
\begin{aligned}
ASC(t^* = T-1|\texttt{BinS}) &= m \times s \times T + \left[\left(T - \frac{T}{2}\right) + \left(T - \frac{3T}{4}\right) + \cdots + \right. \\
&\quad \left. \left(T - \frac{(2^n-1)T}{2^n}\right)\right] \\
&= m \times s \times \left[T + T\left(\frac{1}{2} + \frac{1}{4} + \cdots + \frac{1}{2^n}\right)\right] \\
&= m \times s \times \left[T + T\left(\frac{2^n - 1}{2^n}\right)\right] \\
&\approx m \times s \times 2T
\end{aligned}
$$

3.4.4.1 Average Case. We analyze the average case in Figure 3.9. Binary search makes a decision on two options at each step. Without prior knowledge, two options are equally likely to be selected. Since each option in the previous step is equally likely, all possible decisions are also equally likely in following steps. For example in Figure 3.9, in the third step, four time slices can be dissection point and each of them can be reached with equal probability. Therefore, knowing inference cost for each time slice, we can compute expected inference cost at each step. Computing and summing expected inference cost at each search step gives the expected inference cost of binary search as a function of T , which is equal to $m \times s \times \lceil log_2 T \rceil T/2$.

3.4.5 Line Search. If $UNC(P(\mathcal{L}^T|l^0, l^t))$ is assumed to follow a perfect line as a function of t, then line search can try $t = T - 1$ (the cheapest case) and because $t = 0$ case is already given, can fit a line and pinpoint t^* without any further search. In that case, the total cost of `LineS` is:

$$
\begin{aligned}
ASC(t^*|\texttt{LineS}, perfect\ line) &= IC(T|t=0) + IC(T|t=T-1) \\
&= m \times s \times [(T-0) + (T-(T-1))] \\
&= m \times s \times (T+1)
\end{aligned}
$$

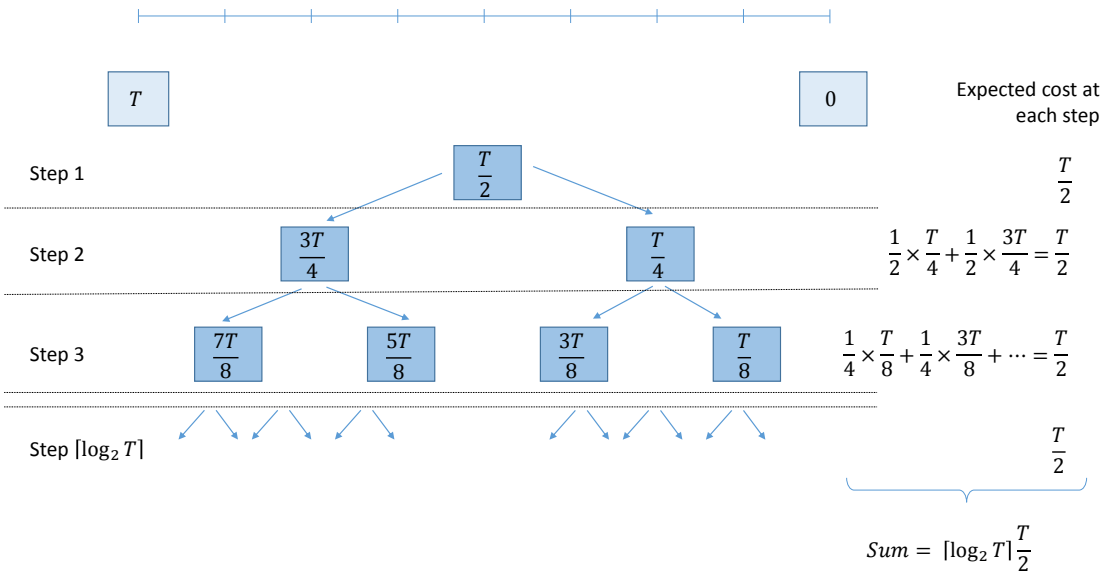


Figure 3.9. Indices of potentially explored time slices by binary search at each step of the search phase. Each of these time slices may or may not be explored. Hence, probability of a time slice for being explored is equal to other time slices of the same row.

As Figure 3.7 shows, the uncertainty is not a perfect line but close to a line. In that case, line search needs to verify that its pinpoint is indeed correct. That is, line search tries $t = T - 1$, $t = t^*$, and $t = t^* - 1$ to verify $UNC(\mathcal{L}^T|l^0, l^{t^*}) < \sigma < UNC(\mathcal{L}^T|l^0, l^{t^*-1})$. Then, for each possible t^*, line search incurs the cost of $IC(T|t = 0), IC(T|t = T - 1), IC(T|t = t^*), IC(T|t = t^* - 1)$.

In order to compute the average case, we need to calculate the summation of

Table 3.2. Aggregated search cost for each t^* value in the case that uncertainty curve is close to a line

t^*	explored slices	aggregated search cost
1	$0, T-1, 1$	$(T-0)+(T-(T-1))+(T-1)$
2	$0, T-1, 1, 2$	$(T-0)+(T-(T-1))+(T-1)+(T-2)$
3	$0, T-1, 2, 3$	$(T-0)+(T-(T-1))+(T-2)+(T-3)$
...	...	...
t	$0, T-1, t-1, t$	$(T-0)+(T-(T-1))+(T-(t-1))$ $+(T-t)$
...	...	...
$T-2$	$0, T-1, T-3, T-2$	$(T-0)+(T-(T-1))+(T-(T-3))$ $+(T-(T-2))$
$T-1$	$0, T-1, T-2$	$(T-0)+(T-(T-1))+(T-(T-2))$

aggregated search cost for each of $t^* = 1, 2, \cdots T-1$, as illustrated in Table 3.2:

$$\begin{aligned}\sum_{t^*=0}^{T-1} ASC(t^*|\texttt{LineS}) &= m \times s \times \big[(T-1)\times(T+1) + [(T-1)+(T-2)+...+2] \\ &\quad + [(T-1)+(T-2)+(T-3)+...+2]\big] \\ &= m \times s \times \left[(T-1)(T+1) + 2 \times \left[\frac{(T-1)T}{2} - 1\right]\right] \\ &= m \times s \times \Big[(T^2-1) + (T-1)T - 2\Big] \\ &= m \times s \times \big(T^2 - 1 + T^2 - T - 2\big) \\ &= m \times s \times \big(2T^2 - T - 3\big)\end{aligned}$$

The average case aggregated search cost is then:

$$\begin{aligned}E[ASC(t^*|\texttt{LineS})] &= \frac{1}{T} \times \sum_{t^*=0}^{T-1} ASC(t^*|\texttt{LineS}) \\ &= \frac{1}{T} \times m \times s \times (2T^2 - T - 3) \\ &= m \times s \times \left(2T - 1 - \frac{3}{T}\right)\end{aligned}$$

3.5 Current Limitations and Generalizations to Other Domains

The DBN model we presented in this article serves as a proof of concept to show the feasibility of using DBNs for vascularization, as has been validated through the similarities of the results with those of agent-based models of vascularization [5], and on an aggregate level through real-world experimental data [2]. The DBN model can be enriched further to reflect reality, such as a 3D instead of a 2D model. Much of these enrichments are engineering problems, where the parents of a node come from a 3D space instead of a 2D space.

The active inference formulation we discussed in this article focused on tissue engineering. Some of the ideas and approaches are, however, general enough to be applicable to several other domains. There are many practical scenarios in which intelligent agents face the question of how much to wait gathering information versus when to act based on what is so far known. This problem is in fact an active inference problem in a temporal domain such as the one we described in this article and hence cost formulations and search results we discussed largely carry over to these problems.

Though our active inference formulation is generic, our inference cost calculations assumed that when an observation is made at time slice t, all variables at that time slice are observed. Hence, expected uncertainty calculations required forward sampling starting from time t to target time T. This meant searching for later time steps incurred less cost (note that `BackS` had less cost than `ForS` for $t^* > 0.3T$ as shown in Equation 3.3). In practice, however, there might be hidden variables or variables that are too costly to acquire so that not all variables can be observed at time t. Then, a simple forward sampling from t to T would not suffice; instead, other approximations algorithms, e.g., likelihood weighting and Gibbs sampling, including all non-observed variables from time 0 to time T need to be performed and hence search for earlier and later steps would incur the same inference cost. In that case, `ForS` and `BackS` would have equal cost when $t^* = T/2$. Still, `BinS` is expected to

outperform `ForS` and `BackS`, and `LineS` is expected to outperform `BinS`, as expected.

3.6 Chapter Conclusion

We presented a dynamic Bayesian network model of vascularization in engineered tissues. The DBN model allows tissue researchers to perform spatial and temporal reasoning for the tissue development process. Additionally, we formulated and evaluated active inference for DBNs in the context of tissue engineering, aimed at answering the question of determining the ideal time to stop a laboratory experiment to guarantee an acceptable uncertainty for the prediction of the future progress of the tissue. We compared several search algorithms and analyzed their inference time complexity, providing closed-form solutions whenever possible. In this article, we focused on the tissue engineering application. However, the active inference formulation for DBNs and the complexity analysis for the search algorithms are general and can potentially be applied to other spatio-temporal domains under some natural assumptions.

CHAPTER 4

ACTIVE INFERENCE FOR DYNAMIC BAYESIAN NETWORK TO REDUCE BATTERY CONSUMPTION IN WIRELESS SENSOR NETWORKS

In this chapter, we will illustrate the second phase of our research framed on a specific real world problem: Battery optimization on wireless sensor networks. We first present an introductory description of the problem followed by our motivation. Then we describe our approach tailored regarding the constraints of the problem. Next, we present our results, and conclude the chapter with a short discussion.

Previously, in Section 2.7, we made a brief introduction to wireless sensor networks, their use cases and battery consumption problem. Wireless sensor networks (WSN) have seen increased deployment mainly due to reduction in hardware costs. Applications of WSNs include monitoring health (of humans, plants, machines, land, etc.), air pollution, water quality, traffic, and detection of movement, fires, and landslides. In majority of the cases, WSNs are not wired to an electric source; they operate on batteries. Even though there have been significant developments in storing more energy in smaller batteries, battery life remains one of the main challenges in effective use of WSNs. Even if batteries are assumed to be cheap, one still has to physically visit each failed sensor to replace the battery. Therefore, several solutions have been proposed to increase battery life, such as reducing communication frequency [3].

A sensor node performs three essential tasks: i) *sensing* in which it converts a physical quantity into a representation, ii) *processing* in which it processes and saves readings, and iii) *communicating* in which it sends and receives data packages. In terms of energy consumption, sensing and processing are negligible compared to communication [3]. Therefore, a typical energy-saving strategy is to communicate only when it is "necessary" and remain silent otherwise. The key questions for this

solution are i) how to determine when it is necessary to communicate and ii) how to handle the missing data that are caused by silent sensors.

One approach is to build a predictive model of the WSN and use the model's predictions in place of the missing data when a sensor is silent. Then, one can wake up the sensor only when the predictive model's predictions are expected to deviate far from the truth. The balance between when to keep the sensor silent and use the predictive model versus when to wake up the sensor and use the sensor readings depends on how much energy one is willing to save versus how accurate one needs the culmination of sensed and predicted readings to be.

Note that the prediction accuracy can be increased by utilizing temporal and spatial correlations among sensors. A sensor's readings are expected to be correlated over time. For example, the current temperature reading of a sensor is expected to be somewhat close to the last temperature reading. Similarly, there might exist correlations across sensors. For example, it is possible that one temperature sensor in one location returns readings that are often a few degrees below compared to another sensor at a different location, and hence this correlation can be utilized to increase prediction accuracy when a sensor is silent but the correlated one is awake.

In this phase of our research, we modeled WSNs as dynamic Gaussian Bayesian networks (dGBn) to exploit the spatio-temporal correlations in WSNs. As baseline models, we used Gaussian Processes (GP) and linear chain graphical model (LC). We also formulated *active inference* problem and solutions: *given an already-trained predictive model and a battery consumption budget, which sensors should communicate their readings and which ones are allowed to remain silent so as to minimize the error over predicted and observed readings?* As baseline active inference methods we tried random selection (RND), sliding window selection (SW), and two variance-based selection methods (maximum variance selection (MV), and iterative maximum

variance selection (iMV)). We proposed a new variance-based active inference method. Finally, we collected temperature data from 50 major airports across the U.S.A. We empirically compared the models and the active inference methods on these two datasets and on an indoor temperature dataset.

4.1 Problem Description and Motivation

Sensor lifespan in a typical wireless sensor network (WSN) depends on battery consumption, and communication is the largest consumer of battery. To increase lifespan, an obvious solution is to reduce the frequency of messages that a sensor sends to other sensors and/or to a central server. The downside of reducing the frequency of messages is that it leads to missing information at various time stamps. One possible remedy is to use a predictive model of the WSN and i) when the sensor reading is available, use the observed reading, and ii) when the sensor reading is missing, predict missing values using the underlying model.

We tackle the following problem: *given a sensor network, a predictive model for the sensor network, θ, and a battery budget consumption budget, B, defined in terms of what percentage of the sensors are allowed to communicate their readings per time step, determine the ideal set of sensors, $\mathcal{S}$, that should communicate their readings to a central server so that the overall error over the network at each time step, $\mathrm{Err}(\mathcal{Y}_t|\mathcal{Y}_t^{\mathcal{S}}, \mathcal{O}, \mathcal{X}_t, \theta)$, is minimized.* More formally, following the notation in Table 4.1, the objective is:

$$\underset{\mathcal{S}}{\operatorname{argmin}} \mathrm{Err}(\mathcal{Y}_t|\mathcal{Y}_t^{\mathcal{S}}, \mathcal{O}, \mathcal{X}_t, \theta) \quad s.t.\ |\mathcal{S}| = B \tag{4.1}$$

Note that in this case, the error is computed over both observed and predicted sensors. The error for the observed sensors can be assumed to be zero or in the case of noisy sensor readings, the error can reflect the noise in the readings. As we discuss

Table 4.1. Notation used in this chapter.

Symbol	Description
t	Current time, for which we run prediction
Y_t^s	Sensor reading for time t and sensor s (if it is communicated to the central server, then it is observed; if not, it is predicted)
$\mathcal{Y}_t$	Sensor readings (including both observed and predicted) of all sensors at time t
$\mathcal{S}$	Set of sensor selected to communicate their readings at the prediction time
X_t^s	Always-observed features of the sensor s at time t, which includes sensor specific information such as ID and location
$\mathcal{X}_t$	Set of observed features of all sensors at time t (i.e, $\mathcal{X}_t$ is the union of X_t^s for all s)
B	Budget, i.e. the number of sensors we can observe at each time
$\mathcal{O}$	Set of observed readings up to time t (i.e., the communicated Y_i^s values for $0 \leq i < t$)
θ	Underlying model
Err	Error function over $\mathcal{Y}$

later in Section 4.3, our goal is to run a WSN at a lower frequency and yet curate a collection of readings (the union of observed and predicted) for a target time t that will be as close as possible to a WSN that runs at full frequency. Hence, our gold standard is the actual sensor readings. This, however, does not preclude one to use a noise filtering approach, like Kalman Filters, on top of our system.

4.1.1 Active Inference. We made an introduction to *active inference* in Section 2.3. Active inference is the technique of selective information gathering for predictions in domains where collective labeling can increase the prediction performance such as network data, or spatio-temporal domains.

In this phase of our research, we model battery optimization in WSNs as an active inference formulation for Bayesian networks. Hence, the Bayesian network formulation allows us to integrate information across sensors and predict missing

readings conditioned on the communicated ones, while the active inference formulation allows us to determine which sensors should communicate their readings so as to maximize the prediction accuracy on the remaining ones, subject to battery consumption constraints.

4.1.2 Approach. We propose to use a dynamic Gaussian Bayesian network (dGBn) to model a WSN. The spatial correlations in the WSN are captured via the edges across sensor variables, while the temporal correlations are captured via the edges between the time slices. During active inference, when a new reading is communicated by a sensor, its value is entered as an observation into the respective variable, and via conditioning on it, its effect is naturally propagated to other variables via the Bayesian network connections. Alternatively, one can choose to use undirected models, such as Markov networks, to model these spatio-temporal correlations. We chose a directed model over an undirected one because, even though the spatial aspect is not necessarily directed, the temporal aspect is naturally directed.

In the proposed dGBn, each sensor s's reading at time t is represented by a Gaussian random variable Y_t^s with the conditional Gaussian distribution $\mathcal{N}(\beta_0 + \boldsymbol{\beta}^T \cdot \boldsymbol{Pa}(Y_t^s), \sigma^2)$ where $\boldsymbol{Pa}(Y_t^s)$ is Y_t^s's parents in the Bayesian network structure, β_0 is the bias, $\boldsymbol{\beta}$ are the edge weights, and σ^2 is the conditional variance. To estimate the structure and the parameters of the dGBn from data, we create an initial 2-slice dynamic Bayesian network with the following structure: the random variables at time t, $\mathcal{Y}_t$ form a fully connected directed-acyclic graph (capturing the spatial correlations) and the temporal connections are added from sensor s's random variable at time $t-1$, Y_{t-1}^s, to itself at time t, Y_t^s. We set all conditional distributions to be linear Gaussian distributions. We estimate the respective β_0, $\boldsymbol{\beta}$, and σ parameters from the training data using L_1 regularization. L_1 regularization naturally leads to a sparse representation by setting some of the β values to exactly zero, in which case the

respective edge is dropped from the structure.

4.1.2.1 Prediction on dGBn. In the battery-saving dGBn model of a WSN, the sensors that communicate their readings are entered as observations while the silent sensors are unobserved variables. We use inference in the dGBn model to predict the readings of the silent sensors, conditioning on the observed readings obtained from the communicating sensors.

Although exact inference methods are intractable [70] on general Bayesian networks (except the most simple structures such as trees), dynamic Bayesian networks with only Gaussian random variables is a special case and allows tractable exact inference [90]. Since we have linear Gaussian distribution for each variable for the first time slice and for intermediate time slices, we can construct a multivariate Gaussian distribution that includes all variables at all time slices from the initial time slice up to the last time slice. To infer conditional Gaussian distribution of each random variable, conditioned on observed variables, we drop all irrelevant variables from the multivariate Gaussian distribution parameters (mean vector and covariance matrix), then we condition the variable to be predicted on observed variables, and insert observed values into the conditional distribution. We next analyze the time complexity of this approach.

Assume the number of sensors is N and our unrolled dGBn has T time slices. Because we will be predicting every variable that is not observed, the first step requires the calculation of the joint multivariate Gaussian distribution, where the covariance matrix's size is $(N \times T)^2$. Next, we need to condition the target variable on all the observations, $\mathcal{O}$. This requires getting the relevant, i.e., $|\mathcal{O}| \times |\mathcal{O}|$, subsection of the covariance matrix, and inverting it. The time complexity of the Gaussian elimination algorithm for matrix inversion is $O(|\mathcal{O}|^3)$. Hence, all operations have polynomial time

complexity.

4.1.2.2 Active Inference: Expected Variance Reduction. In this section, we explain our proposed active inference approach. Our objective, as defined in Equation 4.1, is to find a subset of variables $\mathcal{S}$ to condition at time t to minimize the error over all random variables at time t, $\mathcal{Y}_t$, subject to budgetary constraints. However, clearly, we cannot condition on $Y_t^{\mathcal{S}}$ before the sensors communicate the actual values, and hence, instead of minimizing the actual error, we need to minimize the expected error, where the expectation needs to be taken with respect to $P(Y_t^{\mathcal{S}})$.

$$\underset{\mathcal{S}}{\operatorname{argmin}} \int_{\mathcal{Y}_t^{\mathcal{S}}} \operatorname{Err}(\mathcal{Y}_t|\mathcal{Y}_t^{\mathcal{S}})P(\mathcal{Y}_t^{\mathcal{S}})d\mathcal{Y}_t^{\mathcal{S}} \tag{4.2}$$

For simplicity of notation, we dropped $\mathcal{O}$, $\mathcal{X}_t$, and θ from the conditioning of the error, but note that they are always conditioned on, even though they are not shown in the equations.

The error over all variables at time t, $\mathcal{Y}_t$, is the summation of the errors on the individual variables at time t, Y_t^i for all sensors i:

$$\operatorname{Err}(\mathcal{Y}_t|\mathcal{Y}_t^{\mathcal{S}}) = \sum_{Y_t^i \in \mathcal{Y}_t} \operatorname{Err}(Y_t^i|\mathcal{Y}_t^{\mathcal{S}})$$

where, the individual error is defined as square of the difference between the ground truth Y_t^i and the predicted value $\hat{Y}_t^i$:

$$\operatorname{Err}(Y_t^i|\mathcal{Y}_t^{\mathcal{S}}) \triangleq (Y_t^i - \hat{Y}_t^i|\mathcal{Y}_t^{\mathcal{S}})^2 \tag{4.3}$$

Since we do not know the ground truth Y_t^i, we cannot calculate Equation 4.3. We again calculate the expected error over the distribution of the true reading, $P(Y_t^i)$.

$$\mathbb{E}_{Y_t^i}\big[\operatorname{Err}(Y_t^i|\mathcal{Y}_t^{\mathcal{S}})\big] = \int_{Y_t^i} (Y_t^i - \hat{Y}_t i|\mathcal{Y}_t^{\mathcal{S}})^2 P(Y_t^i)dY_t^i$$

We do not know $P(Y_t^i)$, yet we can approximate this expectation by replacing the true distribution $P(Y_t^i)$ with our estimated distribution, $\hat{P}(Y_t^i|\mathcal{Y}_t^{\mathcal{S}})$.

$$\begin{aligned}\mathbb{E}_{Y_t^i}\big[\,\mathrm{Err}(Y_t^i|\mathcal{Y}_t^{\mathcal{S}})\big] &= \int_{Y_t^i}(Y_t^i - \hat{Y}_t^i|\mathcal{Y}_t^{\mathcal{S}})^2\hat{P}(Y_t^i|\mathcal{Y}_t^{\mathcal{S}})dY_t^i \\ &= VAR(Y_t^i|\mathcal{Y}_t^{\mathcal{S}}) = VAR(Y_t^i|\mathcal{S}) \qquad (4.4)\end{aligned}$$

Note that in the Gaussian case, the predicted value, $\hat{Y}_t^i|\mathcal{Y}_t^{\mathcal{S}}$, is the mean of the estimated distribution, $\hat{P}(Y_t^i|\mathcal{Y}_t^{\mathcal{S}})$. Therefore, the integration in Equation 4.4 is the variance of $\hat{P}(Y_t^i|\mathcal{Y}_t^{\mathcal{S}})$. Since the underlying distributions are Gaussian, the variance does not depend on the actual value of the observations but rather which variables are observed, and hence $VAR(Y_t^i|\mathcal{Y}_t^{\mathcal{S}}) = VAR(Y_t^i|\mathcal{S})$. When we put the expected error back in Equation 4.2, we get:

$$\underset{\mathcal{S}}{\operatorname{argmin}} \int_{\mathcal{Y}_t^{\mathcal{S}}} \Big(\sum_{Y_t^i \in \mathcal{Y}_t} VAR(Y_t^i|\mathcal{S}) \Big) P(\mathcal{Y}_t^{\mathcal{S}}) d\mathcal{Y}_t^{\mathcal{S}}$$

As the variance is independent of $\mathcal{Y}_t^{\mathcal{S}}$, we can move the variance out of the integral:

$$\underset{\mathcal{S}}{\operatorname{argmin}} \Big(\sum_{Y_t^i \in \mathcal{Y}_t} VAR(Y_t^i|\mathcal{S}) \Big) \int_{\mathcal{Y}_t^{\mathcal{S}}} P(\mathcal{Y}_t^{\mathcal{S}}) d\mathcal{Y}_t^{\mathcal{S}}$$

The integral is obviously equal to 1. Therefore our objective function simply reduces to:

$$\mathcal{S}^* = \underset{\mathcal{S} \subset \mathcal{Y}_t}{\operatorname{argmin}} \sum_{Y_t^i \in \mathcal{Y}_t} VAR(Y_t^i|\mathcal{S})$$

Unfortunately finding the optimal $\mathcal{S}^*$ that would lead to minimum expected variance is not tractable. Therefore, to build $\mathcal{S}$ for a target time t, we iteratively select sensors that are expected to minimize the sum of the variances at time t. We call this approach as *expected variance reduction* (EVR).

4.2 Baselines

We will evaluate the performance of both our predictive model, the dynamic Gaussian Bayesian network (dGBn), and our active inference method, the expected variance reduction, (EVR). Therefore we present baselines for our predictive model and for our active inference method.

4.2.1 Predictive Model Baselines.

4.2.1.1 Gaussian Processes. One of the simplest and yet most appropriate strategies for modeling sensor readings over time is Gaussian processes [106]. For WSN, a possible approach is to train one Gaussian process (GP) per sensor using its past readings over time. The advantages are i) training of a GP per sensor and using it for prediction are both remarkably fast, ii) one GP per sensor means it is highly specialized per sensor and can be quite accurate. The disadvantage of using one GP per sensor is that the correlations across sensors are not taken into account. Furthermore, note that during active inference, the model is already trained and deployed and the newly collected data is not utilized to update the model (unless one combines active learning and active inference). Hence, during prediction, a GP model cannot utilize newly observed readings to reduce prediction error on the remaining ones, as the model is static, and the only input to GP is the features $\mathcal{X}$ and not $\mathcal{Y}$, as GP per sensor is not a joint prediction model.

4.2.1.2 Linear Chain Graphical Model. To overcome the downside of GP not being able to incorporate an observed reading at inference time, we propose an alternative baseline model: a linear chain graphical model (LC). In the LC model, we represent each sensor as a separate linear Gaussian Bayesian network. That is, a sensor's reading is a linear Gaussian random variable, of which the unique parent is itself from previous time step. This model utilizes temporal correlations via making use of an earlier observation Y_u^s of the same sensor s in predicting its reading in the current time slice, Y_t^s $(u < t)$. The disadvantage of this model is that it does not

capture spatial correlation. The advantages are: i) it requires much fewer parameters than dGBn, and ii) and exact inference via Kalman filters [65], is much faster than the exact inference for dGBn.

4.2.2 Active Inference Baselines.

4.2.2.1 Random Selection (RND). At each prediction time t, random selection chooses B random sensors for observation. Then, the observed readings for these sensors at time t and all the past observed readings are used during inference to predict the sensor readings for the unobserved ones at time t.

4.2.2.2 Sliding Window Selection (SW). First, we shuffle sensors and fix an order. Then, for prediction at time $t = 0$, we observe the first B sensors. For $t = 1$, we "slide the window" and observe the second set of B sensors, and so on. This is equivalent to a fixed-and-equal frequency schedule.

4.2.2.3 Maximum Variance Selection (MV). For each candidate variable, this method calculates variance conditioned on previous time observations if any. Then it selects top B variables with highest conditional variances. The motivation is similar to our EVR approach, in the sense that the variance is an approximation to error. Therefore, this method always selects variables with highest variance to reduce error. Yet, the major difference with our EVR approach is that MV focuses on high-variance variables, regardless of their impact on other variables, whereas EVR aims to reduce the overall expected variance, and hence takes a variable's expected impact on others.

4.2.2.4 Iterative Maximum Variance Selection (iMV). This is similar to MV, except instead of picking top B variables with highest variance at once, we iteratively pick top variance variable, and pick the next one conditioned on the fact that the previous one is observed, and so on, until B variables are picked. Even though this approach does not consider the expected impact of a variable on others directly, its

impact on others is implicitly taken into account via picking variables iteratively. Still, unlike EVR, this approach does not consider a variable's impact on others when choosing it.

4.2.2.5 Cheating Error Reduction (CER). This strategy's main purpose is to give us an idea about how good the various active inference approaches' results are. As such, this method cheats, and iteratively picks the variable that reduces the actual error the most. Note that, even though this method cheats and picks the best variable, it is still a greedy approach and it is not guaranteed to result in the best set, $\mathcal{S}^*$. Still, the other iterative approaches also have to deal with the greedy nature of the problem, and hence, CER serves as a reasonable lower bound.

4.3 Experimental Evaluation

4.3.1 Data. In this study, we used three datasets derived from two data sources: Intel Research Lab Data [39] and weather data from the Weather Underground website, which we collected.

4.3.1.1 Intel Research Lab Sensor Data. This data consists of temperature (°C), humidity, light, and voltage readings collected from 54 sensors. These sensors, placed in an office environment, were employed for sensing the environment in various frequencies for about 10 days. The data is collected and used by Deshpande et al. [39].

Because LC and dGBn are state-space models, the time stamps in these models are discrete. Therefore, we need to discretize the time stamps in the data as well. We created time bins that consist of 30-minute intervals. We discarded sensors that had missing readings (4 of the 54 sensors were discarded) and when a sensor had more than one reading that fell into one bin, the average reading in that bin is used for the experiment.

On this data, we focused on two types of measurements: temperature and humidity. Using these two types, we created two datasets. For training and testing, we focused on 12pm-6pm time interval, as this interval showed greater diversity in the readings. We used three days of data for training and one day of data for testing. In the end, we obtained 3 days $\times$ 6 hours $\times$ 2 bins/hour = 36 time points, per sensor, for training, and 1 day $\times$ 6 hours $\times$ 2 bins/hour = 12 time points, per sensor, for testing.

4.3.1.2 Weather Underground Data. We fetched temperature (°F) measurements from the Weather Underground website, a weather portal dedicated to forecast and reporting[2]. We collected temperature readings from January 1st, 2015, to March 31st, 2015, for the most crowded airport of each of the 50 states of the U.S.A. Similar to Intel sensor data, we discretized the time of the reading, via creating bins and averaging multiple readings of a sensor (airport) per bin. For this dataset, we used bins of 12am-6am (night), 6am-12pm (morning), 12pm-6pm (afternoon), and 6pm-12am (evening) bins. We used January as our training set and the second week of February as our test set[3]. This resulted in 28 days $\times$ 4 bins/day = 102 time points, per sensor, in the training set and 3 days $\times$ 4 bins/day = 12 time points, per sensor, for the test set

4.3.2 Evaluation Methodology. Our main goal is to have a system that can save battery by putting some sensors to sleep but still is as close as possible to the system that runs on full frequency. That is, the gold standard is the system that runs on full frequency. We therefore computed the error over the entire system, including both predicted *and* observed readings. The observed readings have 0 error because they

[2]https://www.wunderground.com/

[3]We left one-week gap between the training and testing set deliberately to test the models on similar but different patterns

represent the gold standard[4]. Therefore, the system that runs on full frequency has 0 error, albeit terrible battery consumption. Alternatively, a system where all sensors sleep and no sensor communicates their readings would result in the highest error, albeit the perfect (zero) battery consumption. We evaluated our approach and the baselines on several battery budget levels, measured as percentage of sensors that are allowed to communicate their readings at each time step, where the budget ranged from 10% to 50%.

For random selection (RND) and sliding window (SW) active inference approaches, we repeated the experiments 5 times, using different seeds, and report the average errors over trials. The other active inference approaches, i.e., MV, EVR, and CER, do not require us to repeat the experiments as they do not have any variability in their choice of sensors.

On all three models, Gaussian process (GP), linear chain model (LC), and dynamic Gaussian Bayesian network (dGBn), we experimented with random selection (RND), sliding window (SW), and maximum variance selection (MV), which are all baseline active inference approaches. For the dGBn model, we also experimented with iterative maximum variance selection (iMV) and expected variance reduction (EVR). As GP and LC do not incorporate dependencies across sensors, iMV and EVR are equivalent to MV on these models. Finally, we additionally tried the cheating error reduction (CER) method on dGBn, to serve as a reasonable lower bound. Hence we ran experiments with the following model and method combinations: GP with RND, SW, and MV; LC with RND, SW, and MV; and dGBn with RND, SW, MV, iMV,

[4]We are not claiming that the sensor readings are noise-free. What we are targeting is to have a system that can skip some readings to save battery and still produce the same data that would have been produced if it was run on full frequency.

EVR, and CER.

4.3.3 Experimental Results. We calculated both Mean-squared error (MSE) and mean-absolute error (MAE) for our evaluations. For each experiment, we first present MAE then we present MSE results.

We next present the MAE and MSE results on the Intel Temperature, Intel Humidity, and Weather Underground datasets. We first compare the underlying models (Gaussian processes (GP), linear chain graphical model (LC), dynamic Gaussian Bayesian network (dGBn)) when using the common baseline active inference approaches (random selection (RND), sliding window selection (SW), and maximum variance selection (MV)). Then, we fix the model to dynamic Gaussian Bayesian network and show how expected variance reduction (EVR) performs compared to the other active inference approaches.

4.3.3.1 Predictive Model Results. Tables 4.2, 4.3, and 4.4 compare the predictive models by their MAEs, on Intel temperature, Intel humidity, and Weather Underground datasets, respectively. Similarly Tables 4.5, 4.6, and 4.7 compare the predictive models by their MSEs.

All tables have the same layout: rows correspond to underlying models, grouped by baseline active inference approaches. Columns correspond to budgets, ranging from 10% to 50%. All values are MAEs, therefore the lower the better. The lowest MAE in each cell is marked with a bold font.

According to MAE results, for all three datasets, and across all three baseline active inference approaches, dGBn outperform other models on almost all cases (42 out of 45 cases).

The following three tables present the same comparison by MSE results. We obtain a similar aftermath. Out of 45 cases dGBn outperforms the baseline models

Table 4.2. MAE results of predictive models coupled with three baseline active inference approaches on Intel temperature dataset. dGBn performs the best on all cases.

		Budgets				
Methods	Models	10%	20%	30%	40%	45%
RND	GP	1.66	1.47	1.29	1.10	0.92
	LC	1.54	1.28	1.02	0.80	0.62
	dGBn	**0.91**	**0.66**	**0.50**	**0.38**	**0.30**
SW	GP	1.66	1.47	1.29	1.11	0.92
	LC	1.53	1.13	0.81	0.60	0.43
	dGBn	**0.90**	**0.62**	**0.50**	**0.39**	**0.30**
MV	GP	1.75	1.63	1.36	1.23	1.04
	LC	1.44	1.13	0.86	0.68	0.53
	dGBn	**1.39**	**0.91**	**0.66**	**0.46**	**0.33**

Table 4.3. MAE results of predictive models coupled with three baseline active inference approaches on Intel humidity dataset. dGBn performs the best on all cases.

		Budgets				
Methods	Models	10%	20%	30%	40%	45%
RND	GP	3.45	3.07	2.68	2.30	1.92
	LC	3.31	2.66	2.07	1.55	1.18
	dGBn	**2.44**	**1.84**	**1.44**	**1.12**	**0.84**
SW	GP	3.45	3.07	2.70	2.31	1.92
	LC	3.43	2.26	1.53	**1.10**	**0.74**
	dGBn	**2.45**	**1.85**	**1.45**	1.13	0.83
MV	GP	3.58	3.28	2.94	2.51	2.17
	LC	3.73	2.79	2.38	1.84	**1.46**
	dGBn	**1.47**	**2.20**	**1.65**	**1.47**	1.64

in 41 cases. This is not a surprising result, given that dGBn is able to capture both temporal and spatial correlations whereas GP and LC captured only temporal correlations. On the other hand, these results also show that the structure and parameter learning approach we used, as described in Section 4.1.2, is able to learn a

Table 4.4. MAE results of predictive models coupled with three baseline active inference approaches on Weather Underground dataset. dGBn performs the best on all cases.

		Budgets				
Methods	Models	10%	20%	30%	40%	45%
RND	GP	10.10	8.97	7.83	6.67	5.61
	LC	9.06	7.08	5.62	4.41	3.45
	dGBn	**5.21**	**4.00**	**3.11**	**2.50**	**1.88**
SW	GP	10.14	9.01	7.88	6.72	5.62
	LC	8.51	6.18	4.92	3.83	2.84
	dGBn	**5.49**	**4.05**	**3.02**	**2.38**	**1.87**
MV	GP	10.41	9.44	8.59	7.60	6.59
	LC	8.81	6.61	4.91	3.76	2.65
	dGBn	**6.25**	**4.67**	**3.64**	**2.77**	**2.09**

reasonable model that did not overfit or underfit significantly.

Table 4.5. MSE results of predictive models coupled with three baseline active inference approaches on Intel temperature dataset. dGBn performs the best on all cases.

		Budgets				
Methods	Models	10%	20%	30%	40%	45%
RND	GP	4.81	4.25	3.74	3.18	2.69
	LC	4.64	3.72	2.75	1.99	1.49
	dGBn	**2.00**	**1.23**	**0.93**	**0.63**	**0.47**
SW	GP	4.85	4.29	3.80	3.22	2.69
	LC	4.68	3.01	1.77	1.19	0.72
	dGBn	**1.94**	**1.09**	**0.83**	**0.64**	**0.50**
MV	GP	5.26	5.03	4.01	3.81	3.30
	LC	3.77	2.64	1.77	1.29	0.89
	dGBn	**3.60**	**1.93**	**1.14**	**0.71**	**0.42**

Because dGBn had the best performance across the board, next we fix the

Table 4.6. MSE results of predictive models coupled with three baseline active inference approaches on Intel humidity dataset. dGBn performs the best on all cases.

		Budgets				
Methods	Models	10%	20%	30%	40%	45%
RND	GP	18.38	16.34	14.33	12.19	10.21
	LC	18.59	13.91	9.87	6.70	4.73
	dGBn	**12.11**	**7.79**	**5.48**	**4.01**	**2.67**
SW	GP	18.46	16.38	14.46	12.33	10.30
	LC	19.73	9.86	**5.36**	**3.30**	**1.84**
	dGBn	**11.89**	**8.03**	5.49	4.06	2.51
MV	GP	19.65	18.37	16.55	14.07	12.62
	LC	24.02	17.64	13.28	8.97	**6.70**
	dGBn	**10.70**	**9.90**	**7.23**	**7.00**	10.47

Table 4.7. MSE results of predictive models coupled with three baseline active inference approaches on Weather Underground dataset. dGBn performs the best on all cases.

		Budgets				
Methods	Models	10%	20%	30%	40%	45%
RND	GP	151.09	133.99	116.83	99.37	83.96
	LC	139.51	100.99	76.11	56.15	42.38
	dGBn	**51.75**	**37.17**	**26.91**	**20.65**	**14.29**
SW	GP	151.87	135.08	118.07	100.58	84.44
	LC	127.52	82.10	60.45	45.03	30.42
	dGBn	**57.37**	**37.55**	**24.30**	**18.11**	**13.66**
MV	GP	159.34	147.21	137.10	123.43	108.77
	LC	134.57	89.77	58.92	40.61	23.92
	dGBn	**68.25**	**42.85**	**29.87**	**20.81**	**15.30**

model to dGBn and we various active inference approaches.

4.3.3.2 EVR Results. Tables 4.8, 4.9, and 4.10 compare all active inference approaches, i.e., RND, SW, MV, iterative maximum variance selection (iMV), expected variance reduction (EVR), and cheating error reduction (CER) by MAE, on the Intel

Temperature, Intel Humidity, and Weather Underground datasets respectively. The underlying model on all cases is dGBn, as it was shown to be the best performing predictive model. In all cells, the lowest MAE is marked with a bold font, except CER's, as it is a cheating strategy, whose purpose is to provide a reasonable lower bound. All CER scores are greyed out.

These results show that RND and SW are very competitive approaches. MV on the other hand, performs rather poorly in general. This result confirms that focusing on top B high variance variables without taking into account how they will affect other variables is obviously not a good strategy. iMV performs much better because, even though it does not take into account a variable's impact others explicitly, the iterative nature of it prevents us from picking correlated high variance variables. EVR outperformed all other methods on 13 out of 15 cases because it explicitly takes into account the expected impact of a variable on the rest.

In all 15 cases, EVR showed outperformed the other active inference approaches. In fact, EVR achieved very close results to CER, demonstrating EVR's effectiveness.

Table 4.8. MAE of EVR and the baseline active inference approaches on dGBn on Intel temperature dataset. EVR performs the best in all cases.

		Budgets				
		10%	20%	30%	40%	45%
Methods	RND	0.91	0.66	0.50	0.38	0.30
	SW	0.90	0.62	0.50	0.39	0.30
	MV	1.39	0.91	0.66	0.46	0.33
	iMV	0.72	0.55	0.42	0.27	0.31
	EVR	**0.65**	**0.49**	**0.39**	**0.31**	**0.23**
	CER	0.54	0.44	0.34	0.26	0.21

The following tables, 4.11, 4.12, and 4.13, show MSE results to compare active

Table 4.9. MAE of EVR and the baseline active inference approaches on dGBn on Intel humidity dataset. EVR performs the best in all cases.

		Budgets				
		10%	20%	30%	40%	45%
Methods	RND	2.44	1.84	1.44	1.12	0.84
	SW	2.45	1.85	1.45	1.10	0.74
	MV	2.47	2.20	1.65	1.47	1.46
	iMV	2.29	1.39	1.23	**0.76**	**0.63**
	EVR	**1.94**	**1.38**	**0.97**	0.85	**0.63**
	CER	1.66	1.39	1.05	0.76	0.56

Table 4.10. MAE of EVR and the baseline active inference approaches on dGBn on Weather Underground dataset. EVR performs the best in all cases.

		Budgets				
		10%	20%	30%	40%	45%
Methods	RND	5.21	4.00	3.11	2.50	1.88
	SW	5.49	4.05	3.02	2.38	1.87
	MV	6.25	4.67	3.64	2.77	2.09
	iMV	5.78	4.33	2.82	2.28	1.73
	EVR	**4.82**	**3.39**	**2.79**	**2.06**	**1.55**
	CER	3.52	2.67	1.91	1.43	1.05

inference methods on dGBn. These results, similar to MAE results, show that EVR is the best method in 10 cases out of 15. In 1 case, it is slightly better then the closest following method, iMV. In two cases,EVR and iMV tie, being the best method. One interesting result is that on all three datasets, EVR outperformed CER when the budget is 50%. This is simply because CER does not guarantee an optimal solution although it cheats, and EVR, by targeting the overall variance, yields a better error reduction than CER.

4.4 Chapter Conclusion

In this chapter, we tackled the problem of battery consumption of wireless

Table 4.11. MSE of EVR and the baseline active inference approaches on dGBn on Intel temperature dataset. EVR performs the best in all cases.

		Budgets				
		10%	20%	30%	40%	45%
Methods	RND	2.00	1.23	0.93	0.63	0.47
	SW	1.94	1.09	0.83	0.64	0.50
	MV	3.60	1.93	1.14	0.71	0.42
	iMV	1.24	0.80	0.58	0.48	0.38
	EVR	**0.95**	**0.73**	**0.62**	**0.47**	**0.24**
	CER	0.73	0.57	0.51	0.38	0.32

Table 4.12. MSE of EVR and the baseline active inference approaches on dGBn on Intel humidity dataset. EVR performs the best in all cases.

		Budgets				
		10%	20%	30%	40%	45%
Methods	RND	12.11	7.79	5.48	4.01	2.67
	SW	11.89	8.03	5.49	4.06	2.51
	MV	10.70	9.90	7.23	7.00	10.47
	iMV	9.13	**4.37**	4.15	**1.72**	**1.36**
	EVR	**7.72**	4.61	**2.77**	2.36	1.52
	CER	5.61	4.42	3.26	1.99	1.58

Table 4.13. MSE of EVR and the baseline active inference approaches on dGBn on Weather Underground dataset. EVR performs the best in all cases.

		Budgets				
		10%	20%	30%	40%	45%
Methods	RND	51.75	37.17	26.91	20.65	14.29
	SW	57.37	37.55	24.30	18.11	13.66
	MV	68.25	42.85	29.87	20.81	15.30
	iMV	62.01	38.18	**19.64**	15.41	11.03
	EVR	**43.79**	**27.38**	**19.75**	**14.11**	**9.98**
	CER	29.15	20.35	12.57	8.92	5.44

sensor networks (WSNs) that run on full capacity. For this purpose, we proposed a simple solution: At every time step, we will collect readings from a selected subset of sensors, and we will predict the others. This way, we can turn off the sensors that we did not reach out and they can save their batteries. This simple solution raises two important questions: (i) How can we predict the offline sensors with maximum accuracy, possibly using all the past data and all the readings that we collect at the current prediction time? (ii) If we are allowed to select sensors for reading collection at every prediction time step, given the number of sensors to reach out, how can we select the subset that reduces the overall prediction error the most?

For the first question, we proposed a dynamic Gaussian Bayesian network (dGBn), where all readings at each time step are represented as linear Gaussian distributions. We selected two baseline models to evaluate our dGBn model: a Gaussian process (GP) and a linear chain graphical model (LC). dGBn makes use of both spatial and temporal correlations, whilst LC can only use temporal correlations, and GP can use none.

For the second question, we proposed *expected variance reduction* (EVR) method. We compared our method with several baseline methods: random selection (RND), sliding window selection (SW), maximum variance selection (MV), iterative maximum variance selection (iMV), and cheating error reduction selection (CER). Our selection method, EVR, exploits the variance similar to MV and iMV. However, the latter two find the readings with the maximum variance and select them, while our method finds the readings that would yield the maximum change in overall variance once observed.

Our empirical study shows that in the predictive model comparison, dGBn performs the best when any of the three baseline selection models, RND, SW, and MV, is used. In addition, we found that our active inference method, EVR, outperforms all

other methods on dGBn, except CER, which is used to explore the ceiling performance for an active inference method.

CHAPTER 5

ACTIVE INFERENCE FOR FEED-FORWARD NEURAL NETWORKS TO REDUCE BATTERY CONSUMPTION IN WIRELESS SENSOR NETWORKS

In this chapter, we will illustrate the third phase of our research which is a sequel of the previous Chapter 4. Similar to the active inference for dyanmic Bayesian networks (DBNs) for battery saving on wireless sensor networks (WSNs), in this phase, we try to solve battery consumption problem, utilizing a predictive model, and actively collecting information from the domain to boost the model's prediction performance. However, we do not use DBNs. Instead, we use feed-forward neural networks as predictive models, and we develop active inference for feed-forward neural networks.

We first present an introductory description of the problem followed by our motivation. Then we describe our approach tailored regarding the constraints of the problem. Next, we present our results, and conclude the chapter with a short discussion.

5.1 Problem Description and Motivation

In Chapter 1, we shortly discussed the advantages of Dynamic Bayesian networks (DBNs) as predictive models in spatio-temporal domains. First and foremost, they are probabilistic models, and their outputs are probability distributions. One can easily devise uncertainty formulation using estimated probability distributions, if one does not want to straightforwardly utilize the variance of the distribution as the uncertainty. Also, DBNs are multi-label, i.e. multiple output predictive models. Inputs to DBNs are used in predicting all outputs at once, and predictions are not independent. Therefore, true labels provided to one or several outputs can be used

to make predictions on others with lower uncertainty. Another advantage of DBNs is to factorize large joint probabilities into conditional probabilities simplified by conditional independencies. Therefore, learning parameters of a DBN, that is learning conditional distributions are much more affordable than learning joint probabilities.

Nevertheless, DBNs have certain limitations. Unfortunately, even factorizing joint probabilities into conditional probabilities by conditional independencies, do not make inference easy. Typically, exact inference on DBNs is intractable. There are few exceptions, such as linear chains, HMMs, or Gaussian DBNs. Approximate inference techniques, although tractable, are computationally expensive. For example, Gibbs sampling, or more generally Metropolis-Hastings (MH) algorithm are Markov chain Monte Carlo (MCMC) methods, and usually require a long sampling process to converge. Even if one is willing to afford the computational overhead of MCMC methods, then one needs to deal with some meta-parameters of MCMC methods. As we have seen in Chapter 4, MH algorithm has meta-parameters such as the burn-in period, the sample size, and proposal distributions.

To alleviate this problem, we decided to model the domain with linear Gaussian distributions. Notice that Gaussian DBNs allow tractable exact inference. Yet, assuming random variables Gaussian is usually a strong assumption, and inferences may lead to inevitable errors, even though exact inference is used.

Moreover, this solution bears another problem. Note that linear Gaussian distributions are conditional distributions that makes the target variable as the linear combination of its parents. Therefore, this model is limited to domains where variables have linear correlations.

5.2 Approach

As a consequence, by using dynamic Gaussian Bayesian networks (dGBns), we

are limited to represent correlations as linear functions, which yields inevitable errors. Further more, inference is not easy. We might be interested in predictive models that are faster in inference and capable of representing non-linear correlations, such as neural networks.

5.2.1 Neural network modeling. As mentioned in Section 2.4 et seq., first neural networks were proposed as early as in 1950s, in the form of perceptrons [110]. Since then, many architectures have been proposed. In our problem definition, we use feed-forward neural networks (FFNNs) in the following setting:

$$\hat{\boldsymbol{X}}_t = f(\boldsymbol{X}_{t-1}) \tag{5.1}$$

In Equation 5.1, $f(\cdot)$ is our FFNN. $\hat{\boldsymbol{X}}_t$, the output of our FFNN is the vector of random variables that we are trying to predict at the current time. In our WSN setting, they correspond to the sensor readings at time, t. $\boldsymbol{X}_{t-1}$, the input to our FFNN is the vector of random variables at time, $t-1$. In this setting, the predictive model predicts the current time sensor readings based on the previous time sensor readings.

5.2.2 Active Inference for neural networks. In this problem setting, we are trying to predict random variables' current time values using their previous time values, partly observed, partly predicted, and partially observed values at the current time step. Partial observation corresponds to collecting readings from a subset of the sensors, given a budget B. Active inference tries to answer two questions: (i) How to integrate partial observations into predictions? (ii) Which subset should be selected for observation collection? In this chapter, we limit our study to the first question and leave the second question as a future work.

Unlike active inference for DBNs, active inference for neural networks are not

straightforward due to a fundamental problem: On DBNs, we have seen that the true value acquired for a variable at a specific time can be injected into the model as an evidence, and predictions can be updated accordingly. However, on neural networks, this task is not straightforward. As given in Equation 5.1, an output value, X_t^i is a function of any input value X_{t-1}^j, where i and j are not power but indices in their respective vectors. Yet, a feed-forward neural network does not establish a function between output values. Therefore, for all pairs (X_t^i, X_t^j), such that $i \neq j$, we do not have a function $g(\cdot)$ such that $X_t^i = g(X_t^j)$.

One can propose to train a model that can predict X_t^i using a subset of $\boldsymbol{X}_t$. However, there are two essential problems in this approach: (a) based on the observations we acquired for a subset, we can predict X_t^j, but at the same time, our neural network will have a prediction for X_t^j. There is unfortunately no practical way to combine these two distinct predictions. (b) We need virtually infinitely many predictive models to cover all subsets as input. For example, suppose we have N variables. Depending on our budget, we can have the number of subsets, (S) can be in $O(2^N)$. As we will need to train a model per subset, $\mathcal{S}_k$, we will need models in exponential order.

We used two distinct methods to integrate observed values into predicted values. Our first method is Tan and Mayrovouniotis's input training. Our second method is Newton-Meade optimization.

5.2.2.1 Input training. To integrate the observed values into predictions for the unobserved variables at the output, we adopt and extend the input training method proposed by Tan and Mayrovouniotis [135]. In their study, Tan and Mayrovouniotis proposed an autoassociative neural network to reduce the dimensionality of a data set. Their topology was composed of three layers: one input, one hidden layer, and one output. They used the input values as the output values. The hidden layer was smaller

than the input and the output layers, in number of units. After training, they split the model into two: the input and the hidden layer, and the hidden layer and the output layer. They used the first half to project an instance to a compact representation. Then they used the second half to fine tune the compact representation with the original representation.

$$\begin{aligned} \hat{\boldsymbol{X}}_i &= g \circ f(\boldsymbol{X}_i) \\ \hat{\boldsymbol{Y}}_i &= f(\boldsymbol{X}_i) \\ \hat{\boldsymbol{X}}_i &= g(\hat{\boldsymbol{Y}}_i) \end{aligned} \tag{5.2}$$

In Equation 5.2, $g \circ f(\cdot)$ is the autoassociative neural network, $f(\cdot)$ is the first half of the network, $g(\cdot)$ is the second half. $\boldsymbol{X}_i$ is the input to the network, $\hat{\boldsymbol{X}}_i$ is the output of the network, and finally $\hat{\boldsymbol{Y}}_i$ is the output of the hidden layer, therefore the compact representation of the input. Note that on the new instances, the error on $\hat{\boldsymbol{X}}_i$ will be larger than the error on the instances from the training set. The authors conjecture that, the compact representation is not precise and yields this error. A more precise compact representation would have much smaller error.

Inspired by the backpropagation in training of the model, in order to reduce the error on $\hat{\boldsymbol{X}}_i$, they update $\hat{\boldsymbol{Y}}_i$ with backpropagation, as in Equation 5.3. They apply gradient descent to optimize the compact representation.

$$\hat{\boldsymbol{Y}}_i^{+} = \hat{\boldsymbol{Y}}_i + \eta \frac{\partial \operatorname{Err}(\boldsymbol{X}_i, \hat{\boldsymbol{X}}_i)}{\partial \hat{\boldsymbol{Y}}_i} \tag{5.3}$$

Following a similar approach, we used input training to adjust the input to optimize the observed variables at output. We make the assumption that this adjustment will also minimize the error on the unobserved variables. There are several

difference between Tan and Mayrovouniotis's input training and our input training. First, they apply input training on a single layer neural network. In our design, the network can be as complex as needed, as long as backpropagation can be applied. In our experiments, we used a feed-forward neural network model, composed of two hidden layers. Therefore, we were able to apply input training on a more complex model, through hidden layers.

Second, unlike Tan and Mayrovouniotis, in our input training, we do not utilize the error at each output, simply because we do not have ground truth for any output unless it is selected for observation. Therefore, we calculate the error gradient only on the observed subset of the output. Then the gradient of the error is propagated back.

$$\tilde{\boldsymbol{X}}_{t-1}^{+} = \tilde{\boldsymbol{X}}_{t-1} + \eta \frac{\partial \operatorname{Err}(\boldsymbol{X}_t^{*\mathcal{S}_t}, \hat{\boldsymbol{X}}_t^{\mathcal{S}_t})}{\partial \tilde{\boldsymbol{X}}_{t-1}} \tag{5.4}$$

Equation 5.4 formulates input training by gradient descent. $\mathcal{S}_t$ is the set of observed variables at time step, t. The two accents, $\tilde{}$, and $\hat{}$ indicate predicted values and combination of predicted and true values respectively. The superscripts, * and $^+$ indicate the true values and values following the update. Hence, $\tilde{\boldsymbol{X}}_{t-1}$ is the input vector for our model to make prediction for time step, t. This vector is the combination of the predicted values, $\hat{\boldsymbol{X}}_t^{\bar{\mathcal{S}}_t}$ and true value $\boldsymbol{X}_t^{*\mathcal{S}_t}$. Note that the vector of true values here is superscripted with $\bar{\mathcal{S}}_t$, which is the complement of the set of observed variables, i.e. the set of unobserved variables.

In Equation 5.4, $\operatorname{Err}(\boldsymbol{X}_t^{*\mathcal{S}_t}, \hat{\boldsymbol{X}}_t^{\mathcal{S}_t})$ calculates the error over only observed variables by comparing their true values with their predicted values. The gradient of this error is calculated with respect to the input which is the combination of the prediction at the previous time step and the true values of the variables observed at the previous

time step. η is the learning rate. In our experiments, after trying several different values, we selected $\eta = 1$. The number of iterations for gradient descent is 2000.

$$\underset{\boldsymbol{X}_t}{\operatorname{argmin}} \operatorname{Err}(\boldsymbol{X}_t^{*\mathcal{S}_t}, \hat{\boldsymbol{X}}_t^{\mathcal{S}_t}) \approx \underset{\boldsymbol{X}_t}{\operatorname{argmin}} \operatorname{Err}(\boldsymbol{X}_t^{*\bar{\mathcal{S}}_t}, \hat{\boldsymbol{X}}_t^{\bar{\mathcal{S}}_t}) \tag{5.5}$$

Finally, we postulate that adjusting the input variables to minimize the prediction error on observed variables, $\mathcal{S}_t$, will minimize the prediction error on unobserved variables, $\bar{\mathcal{S}}_t$. We formally show this in Equation 5.5.

5.2.2.2 Optimization by Nelder-Mead method. Nelder-Mead, an optimization method proposed by Nelder and Mead in 1965, does not require to calculate the gradient of the function to be optimized, but it proceeds with calculating the function on multiple instances and then makes decision [93]. Details of Nelder-Mead optimization method have been discussed in Appendix A.

5.3 Empirical Evaluation

5.3.1 Error Dissection. In this section we aim to analyze the prediction error by dissecting it into errors of different types. In our problem setting the input is partially observed. We denote it $\tilde{\boldsymbol{X}}_{t-1}$. The error of the prediction of the output, $\operatorname{Err}(\boldsymbol{X}_t^*, \hat{\boldsymbol{X}}_t)$ can be decomposed into two errors:

$$\operatorname{Err}(\boldsymbol{X}_t^*, \hat{\boldsymbol{X}}_t) = \operatorname{Err}_a(\boldsymbol{X}_t^*, \hat{\boldsymbol{X}}_t) + \operatorname{Err}_b(\boldsymbol{X}_t^*, \hat{\boldsymbol{X}}_t) \tag{5.6}$$

5.3.1.1 Error by unobserved input. The first part of the error, Err_a, the accumulating error, stems from the predicted values in the input. Since, the input includes labels predicted in the previous time step, $t-1$, the overall error at the output is partly due to the unobserved values in the input. Because of the sequential predictions, this partial error is not only transferred to future predictions, but also

accumulated by the addition of a new Err_a at each time step.

5.3.2 Model Bias. Yet, the overall error is not limited to the accumulating error, Err_a. It also includes an error generated by the model. We call this error the model bias, and denote Err_b. We conjecture that the model will have an error in the prediction even if it was given a completely observed input.

$$\text{Err}_b(\boldsymbol{X}_t^*, \hat{\boldsymbol{X}}_t) = \text{Err}(\boldsymbol{X}_t^*, f(\boldsymbol{X}_{t-1}^*)) \tag{5.7}$$

Equation 5.7 formally defines the model bias. $\boldsymbol{X}_t^*$ is the vector of true labels at time, t. $f(\cdot)$ represents the predictive model, our neural network in this context. Therefore, $f(\boldsymbol{X}_{t-1}^*)$ is the prediction of the model at time, t. To represent the predictions at time t, we deliberately used $f(\boldsymbol{X}_{t-1}^*)$ to highlight that the prediction is made using true values at the input. An optimal model would have null error. However, in reality, all machine learning models are expected to be suboptimal, which also means that the model is expected to make some error on each instance that was not in the training set.

To explore the existence of the model bias, we ran several experiments on Intel temperature data. In Figure 5.1 and Figure 5.2, we show the existence of the error by model bias via comparing three scenarios. In the first scenario, we predicted 45 time steps. We provided the model with a fully observed input for the first time step prediction. In the following time steps, the model had no observation at all. Then the error of the model is calculated by MAE at each time step. The corresponding curve is represented with blue color and label "no obs". In the second scenario, the same model is used to make prediction for the same time steps. The only difference with the previous scenario is that we provided full observation at the input for each prediction. This satisfies Equation 5.7. In the third scenario, we have the same setting

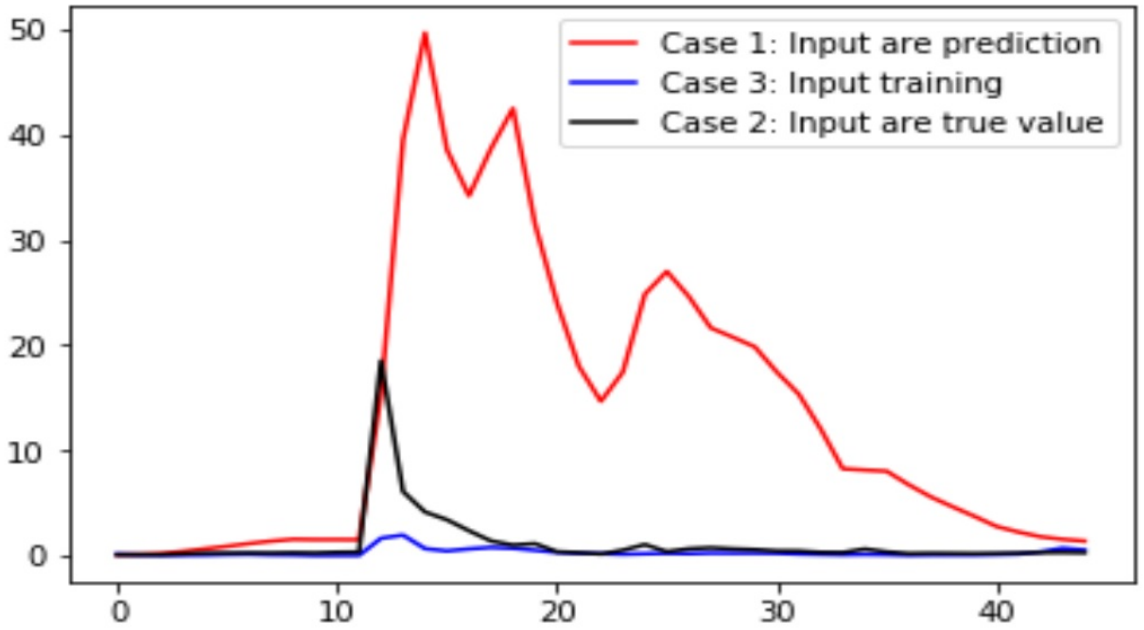


Figure 5.1. MSE along 45 time steps in three scenarios: (i) no observation at input, (ii) the input is fully observed (iii) Nelder-Mead optimization applied on the input using the fully observed output

except that we used our optimization techniques to optimize the input at time step using fully observation at the output.

In Figure 5.1, we used Nelder-Mead optimization for the third scenario. In Figure 5.2, we used input training for the third scenario. Both figures show that when no observation is provided at the input, the model tends to make large error. MSE goes as high as 50, and generally it is above 10. When we provide full observation, maximum MSE obtained is below 20, and generally MSE is below 5, close to 0 most of the time. This comparison shows that the predictions are far from being reliable when no observation is provided. When we provide full observation, the models reduces the error significantly. However, even in that case, the error occurs in most of the cases. Therefore, the model has a bias. The third curve has almost 0 MSE, proving that optimization can reduce the bias of the model.

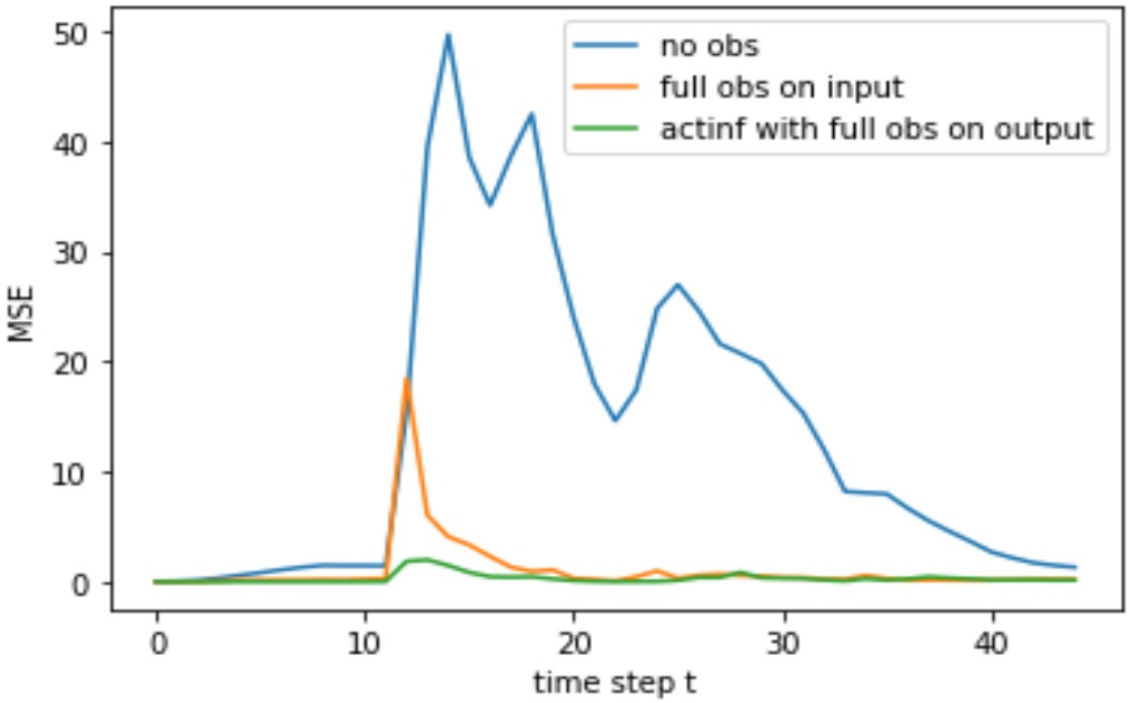


Figure 5.2. MSE along 45 time steps in three scenarios: (i) no observation at input, (ii) the input is fully observed (iii) Input training applied on the input using the fully observed output

Figure 5.3 and Figure 5.4 show the same curves of the second and the third scenarios of Figure 5.1 and Figure 5.2. As the curve for the first scenario does not exist, the scale changed. In this scale, the difference between the second and third scenarios are more visible. In both figures, we can see that the model has bias that appears as the error when the model is fed with full observation. However, the error almost disappears when the input is adjusted to optimize the prediction with respect to the output given as fully observed.

5.3.3 Overfitting of Optimization on Unobserved Output. We use input training or Nelder-Mead optimization method to find the input vector that minimizes the error on unobserved outputs. However, as we do not have values for unobserved outputs, we cannot calculate the error, therefore, we cannot search for the input vector that minimizes the error. Nevertheless, we conjecture that minimizing the

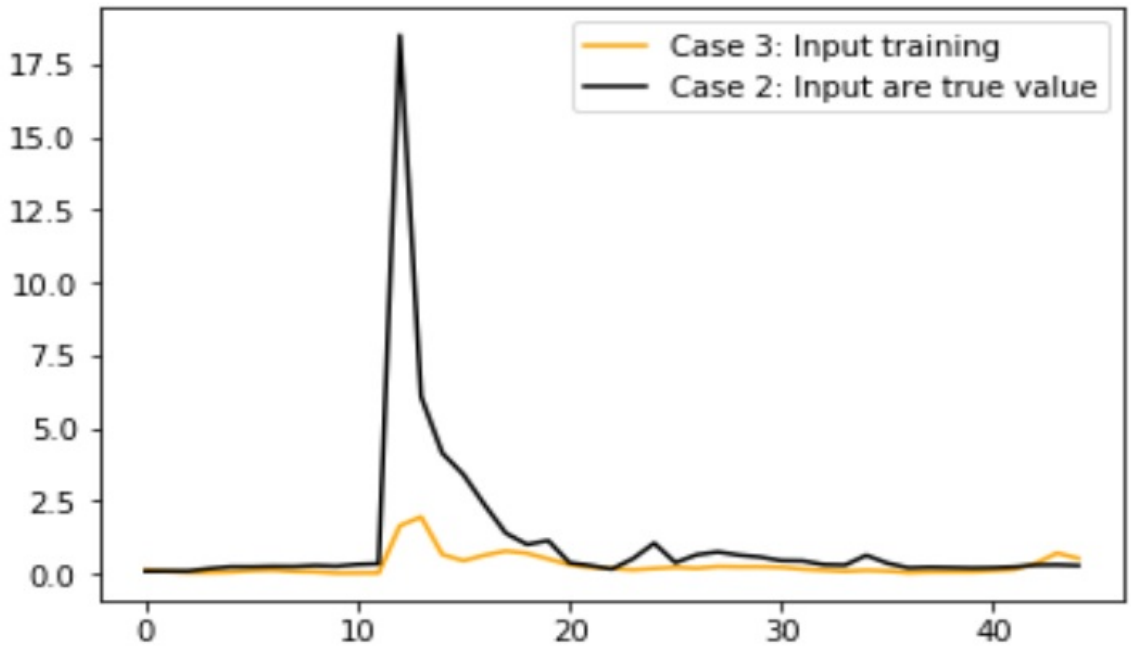


Figure 5.3. MSE along 45 time steps in the last two scenarios of Figure 5.1: (i) the input is fully observed (ii) Nelder-Mead optimization applied on the input using the fully observed output

error on the observed output variables reduces the error on the unobserved output variables. This approach will succeed to the extend it can generalize.

A potential generalization problem may occur in this approach. We optimize the input with respect to the subset of observed variables at the output. However, the target is the unobserved variables. Reducing the error on observed ones by adjusting the input may help reduce the error on the unobserved ones up to a level, then afterwards, it may hurt. This phenomenon is observed in supervised learning. Training to much on a training set can lead to a generalization problem by for example overly complicating the model. The model can then poorly perform on unseen instances. This problem is called overfitting.

Sarle investigates the overfitting on neural networks [118]. He likened this problem to undersmoothing in non-parametric methods, indicating that it may have

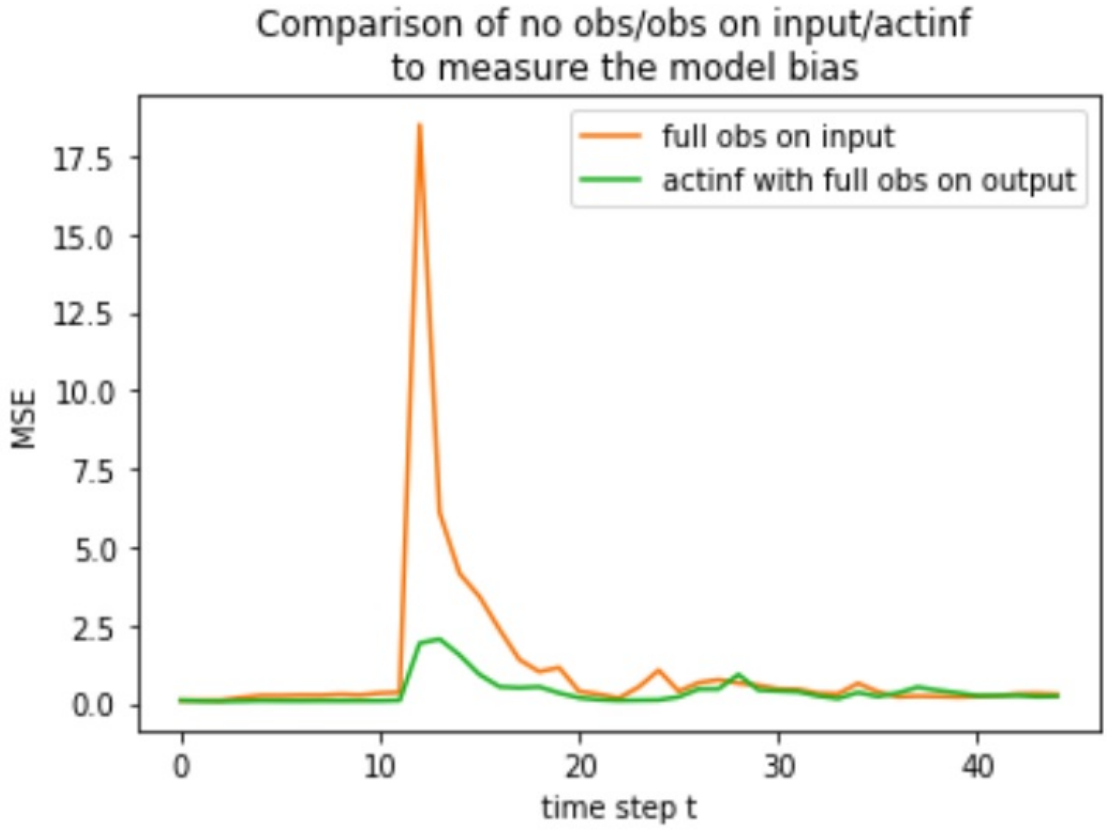


Figure 5.4. MSE along 45 time steps in the last two scenarios of Figure 5.2: (i) the input is fully observed (ii) Input training applied on the input using the fully observed output

more severe consequences such as wild predictions far beyond the range of the data.

In this section, we explore the overfitting problem in optimizing the predictions of the unobserved variables. We run experiments on Intel temperature dataset. We use the previously discussed feed-forward neural network to make predictions at 44 time steps. The first prediction is fed with full observation at the input. Therefore, we do not run optimization for time step, $t = 0$, then we run prediction for the following 44 time steps. We kept the observation rate at 10%, that is we allow five sensors to be observed at each time step. We use random selection to detect sensors to be observed, as described in Chapter 4. Therefore, we run five trials with different seeds. At each time step, t, we used the final prediction from the previous time step,

$t-1$, as the input to the current time prediction. We represent this input as $\tilde{\boldsymbol{X}}_{t-1}$ since it is obtained by combining the predicted values for all variables, $\hat{\boldsymbol{X}}_{t-1}$ and observed values for selected variables, $\mathcal{S}_{t-1}$. Before the final prediction at time step, t, we adjust the input, $\tilde{\boldsymbol{X}}_t$ with respect to the observed variables at the output, $\mathcal{S}_t$. While adjusting the input, we include the observed variables, $\mathcal{S}_{t-1}$, since they do not need to be fixed while predicting values at time step, $t-1$.

We use both aforementioned optimization techniques, input training, and Nelder-Mead optimization. We start by discussing the input training results.

Figure 5.5 shows the overfitting occurring in the time step, $t=5$. In the figure, plots on the left show MSE on the observed output variables, while plots on the right show MSE on the unobserved output variables. Each row of plots corresponds to a trial. We draw MSE on observed and unobserved variables on different xy-planes, simply because their scales are different. Plotting on the same plane squash them into a narrow stripe and make the change much less visible. In each plot, the y-axis represents MSE, and the x-axis represents iterations. We show only the first 200 iterations, though in experiments, the training continues as far as 2000 iterations. The curves on the left starting with a drastic fall quickly becomes flat near 0. This behavior proves that the optimization quickly converges to the true values of the observed output variables.

The curves on the right show the progress of MSE on unobserved output variables for the same iterations. It is interesting to see that in three of the five trials, MSE on unobserved variables decreases very quickly, even more quickly than MSE on observed variables, then starts climbing. In two of these, MSE exceeds the original level of error after climbing. In the third, it almost reaches the original level. In the fourth trial, though MSE does not resume to the original, it increases. Only in the fifth trial, MSE on unobserved variables does not increase. This shows that the

gain on the error reduction on the unobserved variables can easily be reverted with further training after first several iterations, while MSE on the observed variables keep reducing. We interpret this phenomenon as an obvious example of overfitting.

To visualize the overfitting, we plot MSE on unobserved variables with respect to MSE on observed variables for certain time steps, in Figure 5.6. In this figure, rows correspond to three selected time steps in which overfitting is quite visible. Columns correspond to trials. On each plot, x-axis is the MSE on observed variables, y-axis is the MSE on unobserved variables. On each plot, there is one curve which shows the change of MSE on unobserved variables while MSE on observed variables are decreasing. In 13 out of 15 cases, MSE on unobserved variables keeps decreasing up to a point while MSE on observed variables decreases. Yet, at one point, which varies among cases, MSE on unobserved variables starts climbing. In many cases, it climbs to a higher level than its original value. In few cases, it climbs moderately or slightly little, arguably due to the fact that MSE on observed variables does not further decrease.

In a more realistic training design, instead of running the training for a constant number of iterations, we shall do it until the measured error goes below a predefined error threshold, ϵ, and limit the iterations with a predefined number of iterations only if the error never goes below ϵ. Note that we can compare only MSE on observed variables to ϵ. To analyze the existence of overfitting, we shall analyze MSE on unobserved variables for different ϵ values. We tried ϵ between 0 and 0.2, inclusive, with the step size, 0.0005.

In Figure 5.7, we display three selected time steps. The variation of MSEs on both observed and unobserved variables with respect to an error threshold, ϵ are displayed. In all cases, as the threshold is lowered, MSE on observed variables monotonically decreases. However, in nine cases, MSE on unobserved variables first

decreases then starts climbing. According to these results, nine cases show explicit overfitting. We refer the reader to Appendix B to examine all 44 time steps.

These plots, which are organized in different aspects, reveals that our input training optimization can overfit and considerably hinder the prediction performance.

5.3.4 Regularizing Optimization against Overfitting. There are a few potential remedies for overfitting. Arguably the most crude approach is to set a lower iteration number to stop the gradient descent search in input training, or the simplex search in Nelder-Mead optimization. Yet, this approach is a blind technique that may cause some undesirable consequences on both directions of the underfitting-overfitting route. In other words, a lower limit for iteration can usually result an underfitting. Moreover, there is no guarantee that the optimization will not occasionally overfit.

A second approach can be to split observed variables, $\mathcal{S}_t$ into two distinct subsets, $\mathcal{P}_t$ and $\mathcal{V}_t$ then using $\mathcal{P}_t$ for optimization target and $\mathcal{V}_t$ as the reference for validation, as in Equation 5.8 and in Equation 5.9.

$$\mathcal{S}_t = \mathcal{P}_t \cup \mathcal{V}_t \tag{5.8}$$

$$\mathcal{P}_t \cap \mathcal{V}_t = \emptyset \tag{5.9}$$

We conjecture that if there exists a correlation between $\mathcal{S}_t$ and $\bar{\mathcal{S}}_t$ that the optimization can exploit to reduce the prediction error on unobserved variables, $\bar{\mathcal{S}}_t$ by fitting the input in a way that the prediction error on observed variables, $\mathcal{S}_t$, are minimized, then a similar correlation can exist between the subset of observed variables referred for validation, $\mathcal{V}_t$, and the set of unobserved variables, $\bar{\mathcal{S}}_t$.

$$\underset{\boldsymbol{X}_t}{\operatorname{argmin}} \operatorname{Err}(\boldsymbol{X}_t^{*\mathcal{P}_t}, \hat{\boldsymbol{X}}_t^{\mathcal{P}_t}) \approx \underset{\boldsymbol{X}_t}{\operatorname{argmin}} \operatorname{Err}(\boldsymbol{X}_t^{*\mathcal{V}_t}, \hat{\boldsymbol{X}}_t^{\mathcal{V}_t}) \tag{5.10}$$

Equation 5.10 is similar to Equation 5.5 in the sense that optimizing predictions with respect to the subset, $\mathcal{P}_t$, of observed variables, $\mathcal{S}_t$ approximates the optimization of the subset, $\mathcal{V}_t$, of the observed variables. Here, $\mathcal{P}_t$ represents the subset of observed variables that are used in the optimization as the target in minimizing the error. $\mathcal{V}_t$ is the subset of observed variables that are used measure the error. As this subset is observed, the error can indeed be calculated in parallel through optimization. The error on these observed variables, $\text{Err}(\boldsymbol{X}_t^{*\mathcal{V}_t}, \hat{\boldsymbol{X}}_t^{\mathcal{V}_t})$, is expected to reduce up to a point, then to start increasing afterwards.

$$\underset{\boldsymbol{X}_t}{\text{argmin}}\, \text{Err}(\boldsymbol{X}_t^{*\mathcal{V}_t}, \hat{\boldsymbol{X}}_t^{\mathcal{V}_t}) \approx \underset{\boldsymbol{X}_t}{\text{argmin}}\, \text{Err}(\boldsymbol{X}_t^{*\bar{\mathcal{S}}_t}, \hat{\boldsymbol{X}}_t^{\bar{\mathcal{S}}_t}) \tag{5.11}$$

Similar to our postulate described in Equation 5.10, we postulate that minimizing the error on a subset of observed variables, will also minimize the error on unobserved variables, as mathematically formulated in Equation 5.11. In this equation, $\mathcal{V}_t$ represents observed variables spared for validation through optimization process, and $\bar{\mathcal{S}}_t$ represents all unobserved variables.

We conducted a set of experiments to investigate whether our postulate holds. On Intel temperature data set, using FFNN, we ran sequential prediction over 44 time steps, starting with a complete observation, and 30% observation at each time step. We used random selection. At each time step, we split observed variables into two sets, $\mathcal{P}_t$ and $\mathcal{V}_t$, as described above. As we used random sampling, we ran 5 trials.

In Figure 5.8, we show plots for three selected time steps, 7, 19, 29. In this figure, each row corresponds to a time step, each column corresponds to a trial. The y-axis is MSE, the x-axis is the error threshold, ϵ. In each plot, the magenta curve is MSE on validation set, $\mathcal{V}_t$, the blue curve is MSE on optimization set, $\mathcal{P}_t$. In 14 out of 15 cases exhibited in the figure, MSE on $\mathcal{V}_t$ goes down so long as we reduce ϵ. Yet, it

starts climbing if the optimization continues to reduce the error on $\mathcal{P}_t$. Therefore, we can use half of the observed subset for validation, and thus we can prevent overfitting.

In Appendix C, we share the same plots as in Figure 5.8 for all 44 time steps.

5.4 Chapter Conclusion

In this chapter, we first discussed the drawbacks of dynamic Bayesian networks in sequential prediction. Then, we proposed neural networks as a surrogate predictive model addressing the disadvantages of dynamic Bayesian network models. Next, we tackled the problem of observation integration into predictions in neural networks. We proposed two approaches to integrating true values acquired from variables observed at the current time step, which is also the time step that we make prediction. Both methods aim to minimize the error on observed variables by adjusting the input. For both approaches, we assume that input values that minimize the error on observed variables will also converge the error on unobserved variables to minimum.

The first method is input training. We used the backpropagation to update the input values unlike the conventional backpropagation which is used to update the model parameters.

The second method is Nelder-Mead optimization algorithm. We confirmed that problems that we saw in active inference using input training also exist in active inference using Nelder-Mead algorithm.

We first address model bias in the overall error. We show that the model bias exists and can be reduced to zero or near zero using input training or Nelder-Mead algorighm. Then we showed that overfitting can hurt active inference. We first empirically showed the existence of overfitting. Then we proposed a method to deal with overfitting.

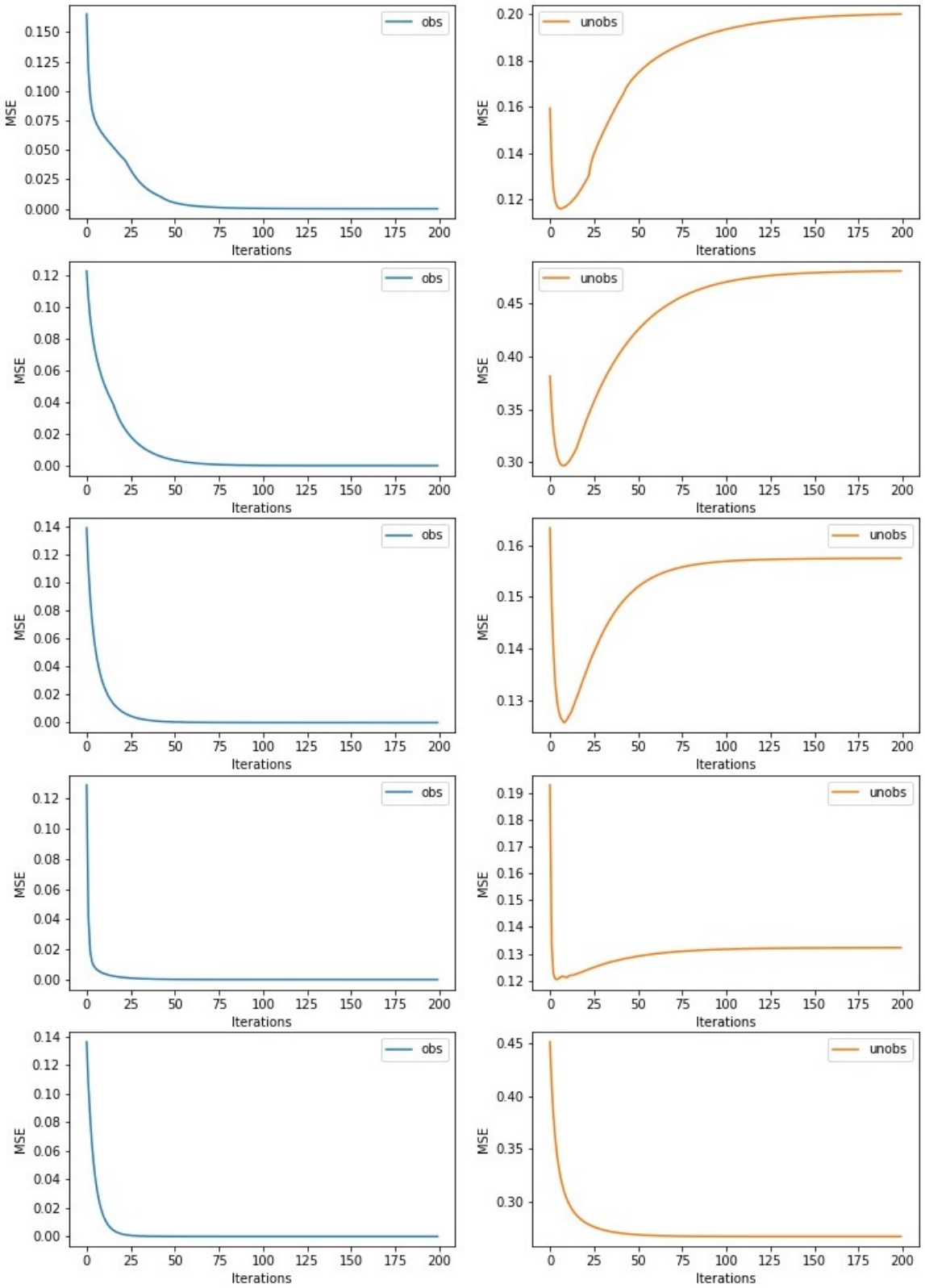


Figure 5.5. MSEs on observed (blue) and unobserved (orange) variables for the first 200 iterations of gradient descent in input training. Each row corresponds to a trial.

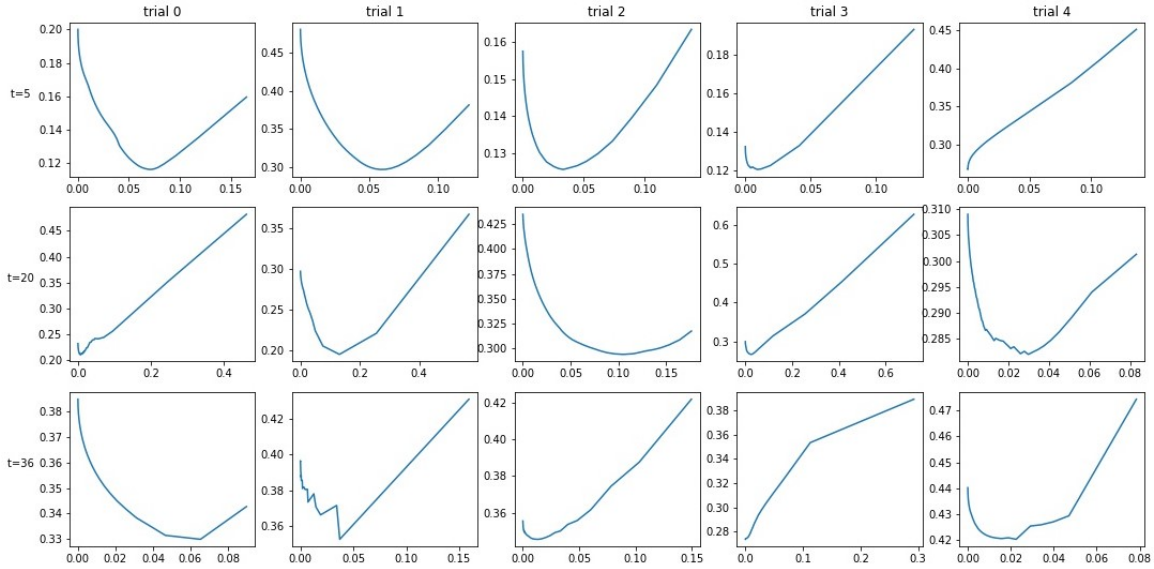


Figure 5.6. MSE on unobserved variables with respect to MSE on observed variables. X-axis is MSE on observed variables, and y-axis is MSE on unobserverd variables. Each row corresponds to a selected time step. Each column corresponds to a trial.

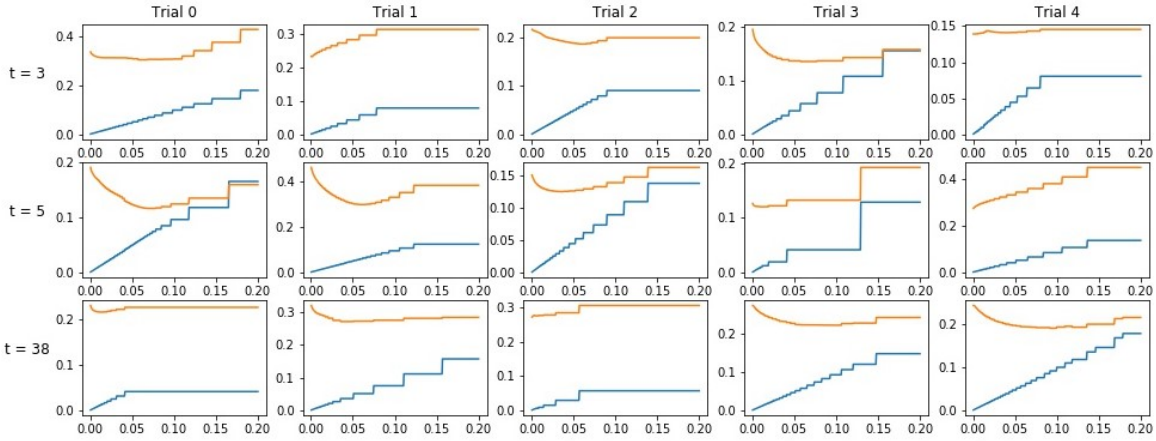


Figure 5.7. MSE on observed (blue) and on unobserved (orange) variables. X-axis is the error threshold, ϵ, set for the termination of gradient descent.

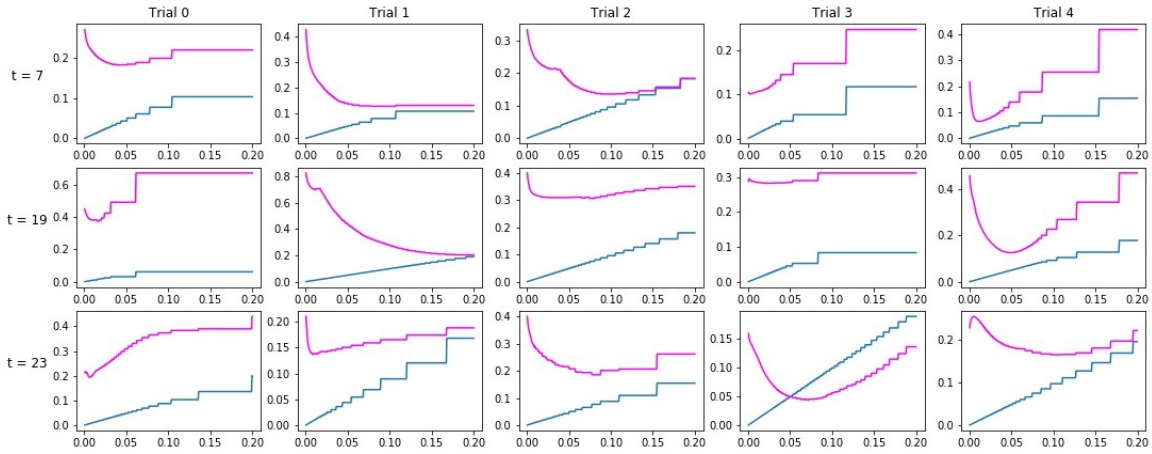


Figure 5.8. MSE on observed variables that are spared for optimization (blue) and on observed variables that are spared for validation (magenta). X-axis is the error threshold, ϵ, set for the termination of gradient descent.

CHAPTER 6

CONCLUSION

In this research, I explored *active inference* for predictive models of spatio-temporal domains. As spatio-temporal domains, I investigated two domains: tissue engineering and wireless sensor networks. In both domains, I developed predictive models, and to be used in conjunction with models, I proposed active inference methods.

Active inference helps a predictive model to make better predictions, in prediction time. There are a few prerequisites to enable active inference help the predictive model. The first and foremost, active inference optimizes information acquisition. In this research I narrowed down information acquisition to label acquisition. Therefore, one can use active inference only when information acquisition is allowed during prediction. In many machine learning applications, once the model is deployed for inference/prediction, information acquisition is not allowed or simply not possible. The second prerequisite is that the domain should support correlations between instance labels. Thereby, being aware of labels of related instances, a predictive model can potentially make a more accurate prediction for the label of an instance. The third prerequisite is that the predictive model should be capable of exploiting the correlations during prediction.

In three different chapters, Chapter 3, Chapter 4, and Chapter 5, I described the active inference methods that I developed based on the predictive model requirements and domain mechanics. In each phase, I first defined domain variables that need to be predicted, based on the problem definition. Then I explored spatio-temporal correlations existing in the domain. Next, I proposed a predictive model that can

utilize those correlations in predictions. Considering the model capabilities, I developed an active inference method that targets the prediction uncertainty. Finally, with empirical studies, I showed that the proposed active inference method increases the prediction performance of the predictive model.

6.1 Contributions

This research has three phases: (i) Active inference for dynamic Bayesian networks in an application of tissue engineering, (ii) Active inference for dynamic Bayesian networks for battery saving in wireless sensor networks, and (iii) Active inference for neural networks for battery saving in wireless sensor networks.

In the first phase, which is described in Chapter 3, I explored the tissue engineering domain for the purpose of vascularization prediction. The problem that I attempted to solve is to find out blood vessel development probabilities for atomic regions in a tissue development site. After gridding the site into regions, I modeled the regions in a dynamic Bayesian network, which then I used to predict the probability of being invaded by blood vessels at each time step. Using this model, one can predict the outcome of a wet tissue engineering experiment in terms of blood vessel development in the tissue development site.

Following the modeling step, I focused on the problem of finding the optimal time to stop the wet experiment. For this purpose, I proposed a new formulation of uncertainty of the prediction of the blood vessel development at the finale of the experiment. This uncertainty formulation is a function of evidence, in the form of observation, provided to the DBN model at a certain time after the experiment has started. I showed that the uncertainty depends not only on the observed values, but also on the time through the process of the experiment. Then I explored the trade-off of stopping the wet experiment early and making observation to update the model's

prediction about the outcome with a high uncertainty versus stopping the experiment later to reduce prediction uncertainty but paying higher costs of experimenting and losing valuable time while waiting.

Finally, based on the uncertainty formulation and considering the trade-off, I tried to detect the optimal time to stop a wet experiment given the highest uncertainty level that is affordable, and initial experiment observations. I compared the results with simulations, and actual wet experiments results, and proved the validity of my model.

In the second phase of my research, which is described in Chapter 4, I focused on battery consumption problem in wireless sensor networks. I proposed a simple yet effective solution: Given a wireless sensor network, one can use a predictive model to predict readings for some sensors, rather then collecting readings from all of them. This way, one can save energy on those that are kept silent. I tried to answer two questions on this approach: (i) Which predictive model can one use to predict readings? (ii) Given an overall saving, which readings should one collect at each time step to increase prediction performance?

For the first question, I proposed a dynamic Bayesian network in which all readings are represented as linear Gaussian distributions. The topology was learned by L1 regularization and the parameters are learned by maximum likelihood estimation. For the second question, after exploring some generic variance-based active inference methods, I developed expected variance reduction method. On an empirical study, I showed that using baseline active inference methods, the DBN model outperforms the baseline predictive models by exploiting both spatial and temporal correlations. Also, I showed that on DBN, the proposed active inference method, the expected variance reduction, outperforms all baseline active inference methods. I conducted the empirical study on three data sets, one of which I scraped from the

web and preprocessed.

In the third phase of my research, as described in Chapter 5, I focused on certain weaknesses of DBNs modeling WSNs. The complexity of inference, the requirement of linearity on the correlations given Gaussian assumption are some issues to name. In the same domain as the second phase, to replace DBNs, I proposed using neural networks. I trained a FFNN to predict sensor readings. The choice of using FFNN, while answering the first research question in the second phase, bears a follow-up question: How can one integrate observations into predictions in a FFNN, where the output is the current time readings and the input is the previous time readings? I proposed to use input training in conditioning the input to match the predictions to the observed values, based on the assumption that matching a subset of variables at the output will match the remaining outputs. I compared the performance of input training with the performance of a well known optimization method, Nelder-Mead.

In empirical studies, I noticed two problems: (i) the model bias, and (ii) the overfitting of optimization. Input training, as well as other optimization methods can overcome the error due to the model bias. For the overfitting problem, I proposed splitting observations into two subsets: target subset, and validation subset. Input training can be controlled through watching error progress on the validation set, while optimizing on the target set. I empirically showed that this approach can solve the overfitting problem.

6.2 Future Directions

This research can be extended in a few different directions. I discuss these directions next.

6.2.1 Integrating Reinforcement Learning with Active Inference. In the second and third phases of this research, the goal was to find the most useful variables

for the prediction of other variables' values at the current time. However, one might easily face a problem where the objective can include benefits of active inference not only at the current time but also in the future. For such a specification, decision-making of the current time should take into account succeeding decision-makings. Therefore, the uncertainty of decision-makings, which can also be named *actions*, and contributions following the actions, which can also be named *rewards* should be considered as part of the current time decision-making. Once the active inference is formulated this way, many reinforcement learning techniques can easily be applied, such as temporal difference learning.

6.2.2 Active Inference For Human Contribution. Active inference for predictive models of spatio-temporal domains will have a greater impact in domains where information collection is more costly. For example, in a domain where human contribution, such as crowd-sourcing, is required, the performance of active inference will become paramount.

One such domain is social media. Some objectives related to social media data require human effort, for example, assessing political inclination on social media. Suppose that one aims at devising regional public opinion on a certain political topic. In the case that we have such multiple regions, and that we would like to track the trend of public opinion through a certain period of time, it will be quite costly concentrating human effort directly on the social media data to devise public opinion. We may wish to use a predictive model that can exploit spatio-temporal correlations existing between regions and also that can incorporate collected evidence into its predictions. Thereby, we can design an active inference method that will collaborate with this predictive model, that will account the cost of human effort, and the expected contribution of acquiring actual public opinion from each of the regions with the help of human analysis, and that can finally select the most useful locations

to concentrate the human effort on.

6.2.3 Transparent and Interactive Inference. In the previous future direction, we discussed the potential impact of active inference in domains where human contribution increases information gathering costs and introduces arguably more complex constraints.

As a further stage, one can make use of human contribution not only in information acquisition, such as labeling in this case, but also to make active inference more efficient, by rendering human decision-making part of the active inference decision-making process. The first step in this direction is providing an interface to the human to make the decision-making of active inference more transparent. For example in the domain of wireless sensor networks, considering that the goal is reducing the battery usage, the human can decide to focus on some specific sensors in certain time steps after seeing that those sensors are being omitted for a long time. Therefore, the interface should display which sensors have been chosen in the past and which sensors are selected for the current prediction. The interface should also inform the user about the reason behind the selection of the sensors to be accessed.

After transparency, the following step is rendering active inference interactive. The goal here is to make the user more involved into the decision-making process of selecting information sources for observation. In the domain of wireless sensor networks, this translates to providing the human with an active role in the decision of which sensors should be selected for retrieving their readings.

APPENDIX A

FORMAL DESCRIPTION OF NELDER-MEAD OPTIMIZATION METHOD

In a n dimensional search space, it always keeps $n+1$ instances. It iteratively searches the optima through the search space. At each iteration, it evaluates the current status, and then proceeds with one of the four options: (i) reflection, (ii) expansion, (iii) contraction, and (iv) shrink. For each option, the algorithm uses a separate parameter.

Suppose we are minimizing a function, $f(\cdot)$. We start off by selecting a simplex, which is a set of $n+1$ instances from the n dimensional search space. Then we start search loop. At each iteration of the loop, the algorithm starts with ordering initially selected $n+1$ instances: $f(x_1) < f(x_2) < \cdots < f(x_{n+1})$. Then calculates the centroid, x_c of all instances except x_{n+1}, as this instance resulted the worst value of the function among all instances. Then we *reflect* x_{n+1} with respect to x_0, and fix the point in the space, namely point x_r. It is calculated as:

$$x_r = x_0 + \alpha(x_0 - x_{n+1}) \tag{A.1}$$

where α is the step size for the reflection, and it is usually set to 1. **Reflection:** Next, we compare $f(x_r)$ with the best point, x_1, the second worst point, x_n, and the worst point, x_{n+1}. If $f(x_1) < f(x_r) < f(x_n)$, that is, if the reflection point is worse than the best point in the current simplex, but better than the second worst, then we discard the worst point, x_{n+1} from the simplex and include the reflection point, x_r. **Expansion:** If this case does not hold, then we check whether $f(x_r) < f(x_1)$ holds, that is if the reflection point is better than the best point, the reflection is expanded, that is the reflection point is dragged further away from the centroid. Then the new point is included to the simplex and the worst point is discarded. The expansion point, x_e is calculated as following:

$$x_e = x_0 + \gamma(x_r - x_0) \tag{A.2}$$

Contraction: If that is not the case either, then $f(x_r) > f(x_n)$, that is the reflection point is worse than the second worst. We compare the reflection point, x_r, with the worst point, x_{n+1}. If $f(x_r) < f(x_{n+1})$, that is the reflection point is better than the worst point, we contract the reflection point, that is we drag the reflection point towards the centroid.

$$x_c = x_0 + \rho(x_{n+1} - x_0) \tag{A.3}$$

Shrinking: Finally, if the reflection point is worse than the worst point of the simplex, then we shrink the simplex towards the best point. That is, we update each point in the simplex except the best point, following this equation:

$$x_i = x_i + \sigma(x_i - x_1) \tag{A.4}$$

In these equations, $\alpha, \gamma, \rho, \sigma$ are update coefficients. Usually, their values are set such that $\alpha = 1, \gamma = 2, \rho = 0.5, \sigma = 0.5$

APPENDIX B

OVERFITTING RESULTS

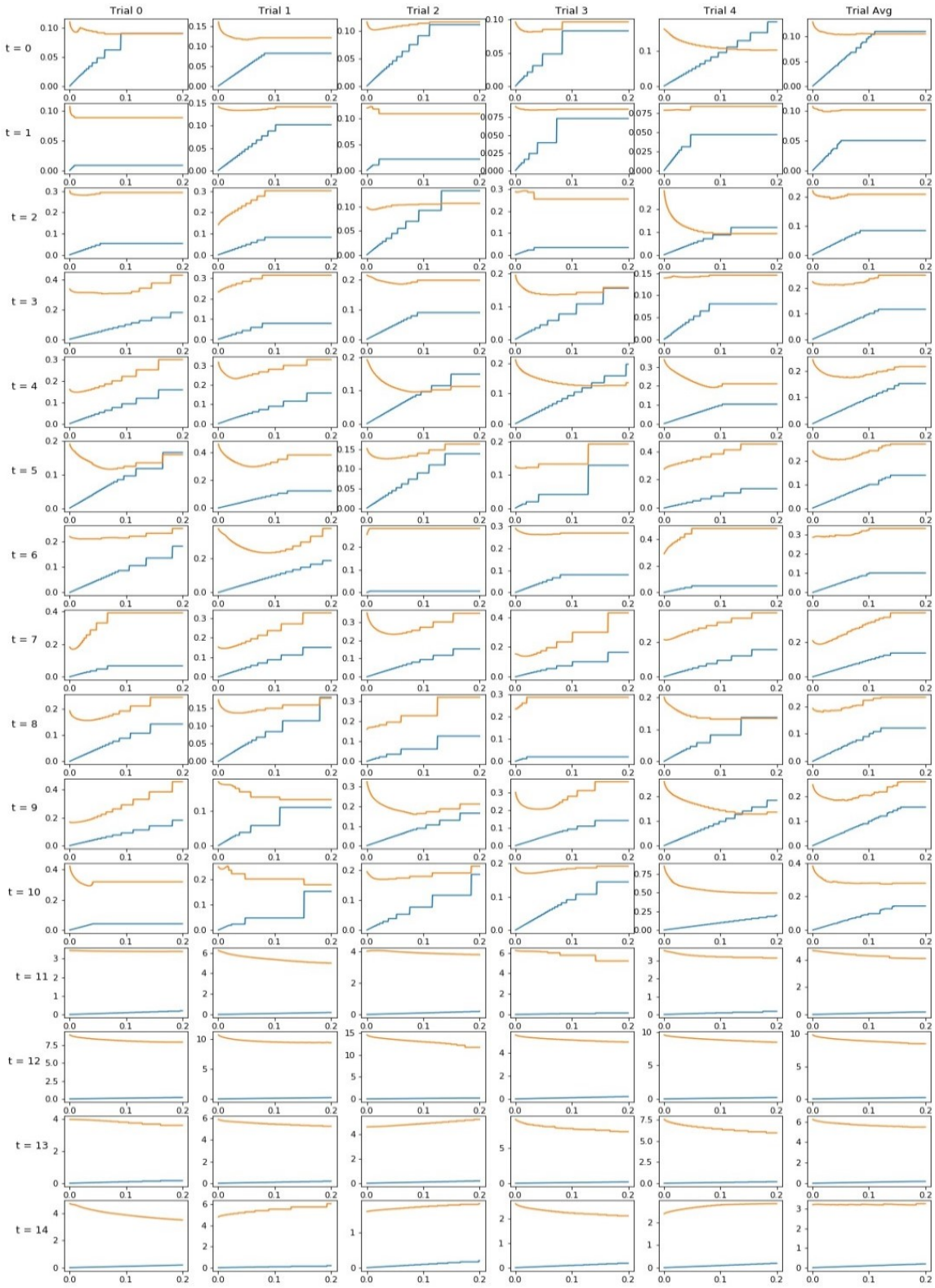


Figure B.1. MSE on observed variables (blue) and MSE on unobserved variables (MSE) over the first 15 time steps with respect to ϵ. Each row corresponds to a time step, each column corresponds to a trial.

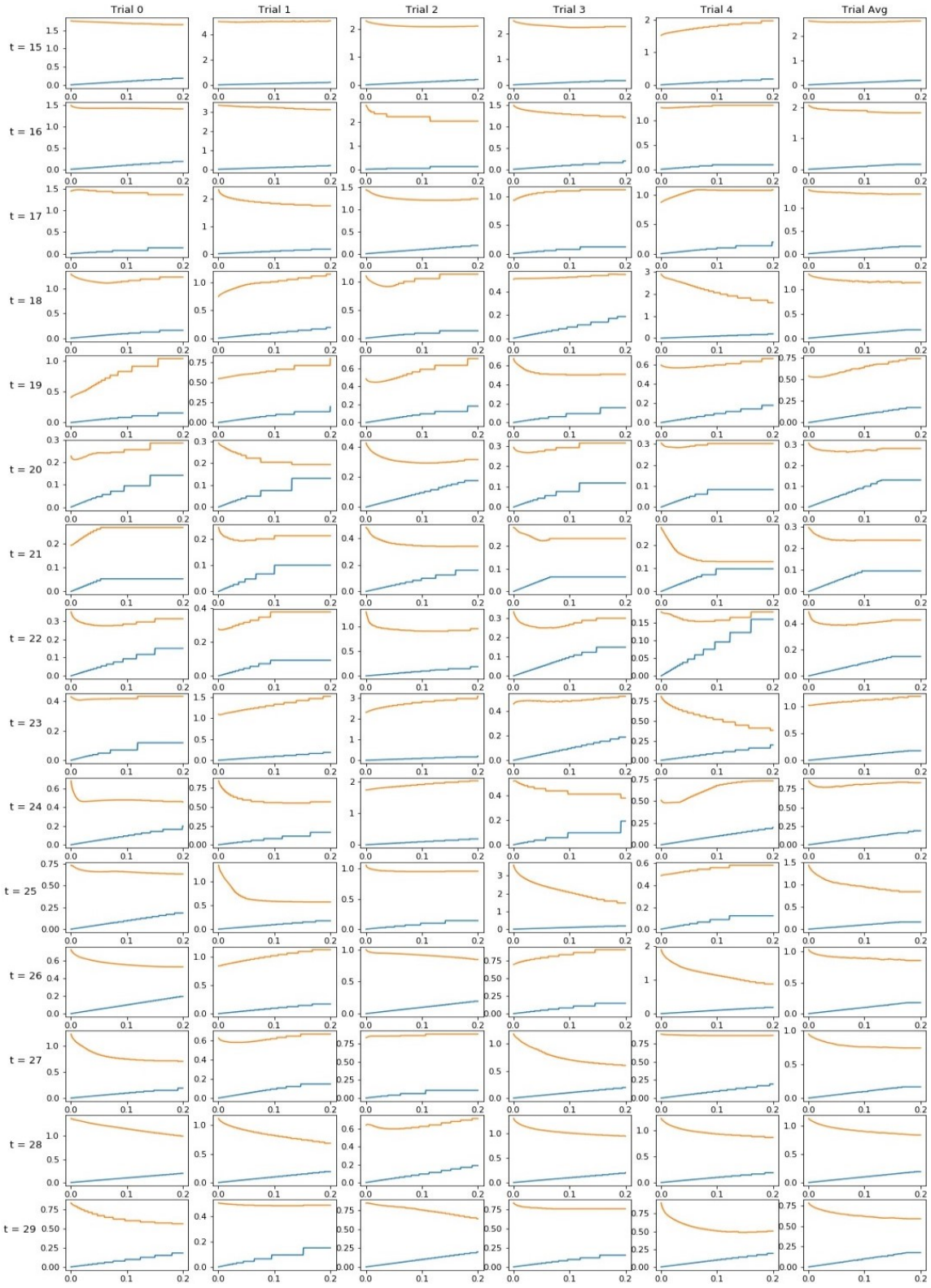


Figure B.2. MSE on observed variables (blue) and MSE on unobserved variables (MSE) over the second 15 time steps with respect to ϵ. Each row corresponds to a time step, each column corresponds to a trial.

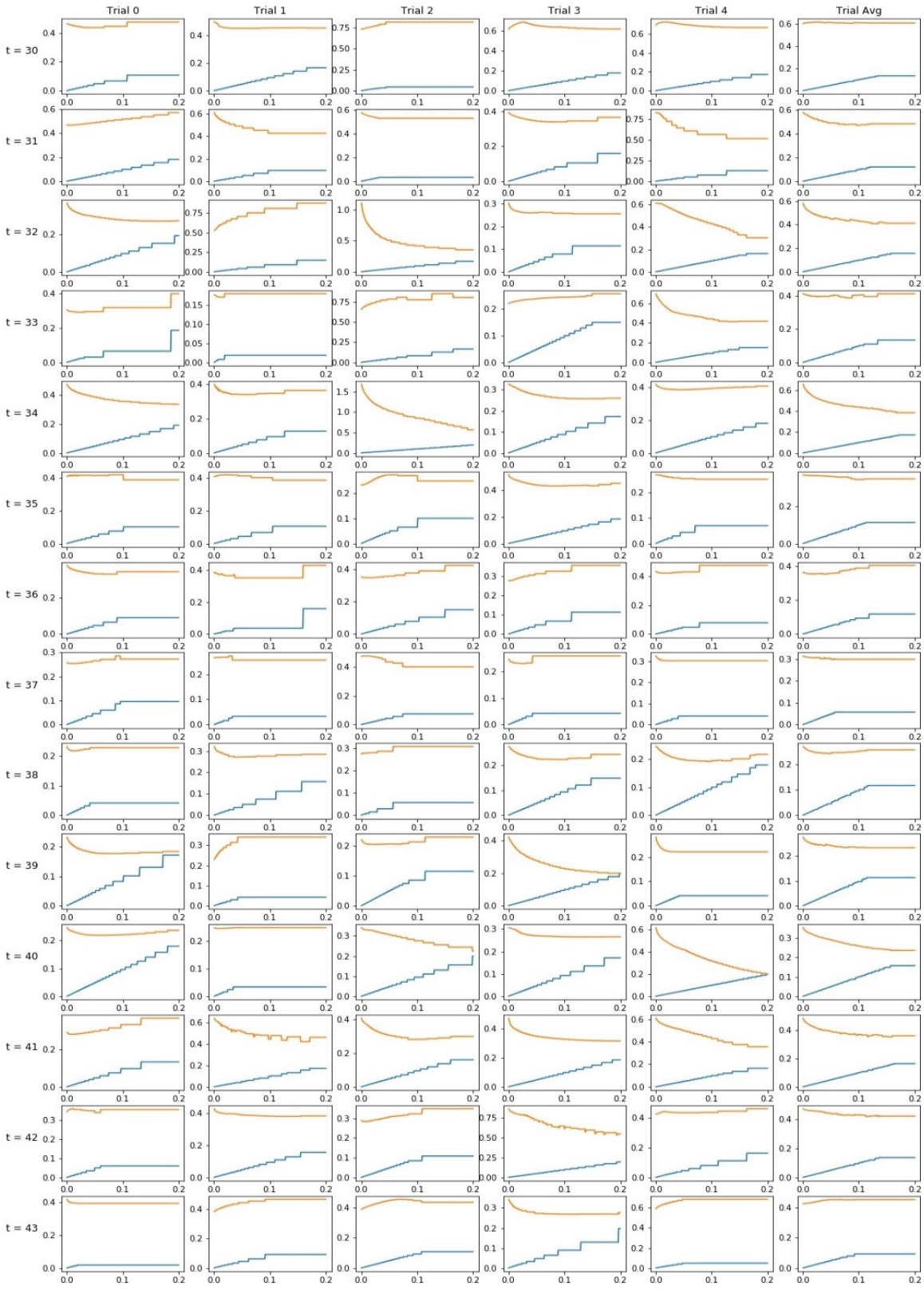


Figure B.3. MSE on observed variables (blue) and MSE on unobserved variables (MSE) over the last 14 time steps with respect to ϵ. Each row corresponds to a time step, each column corresponds to a trial.

APPENDIX C

OVERFITTING REGULARIZATION RESULTS

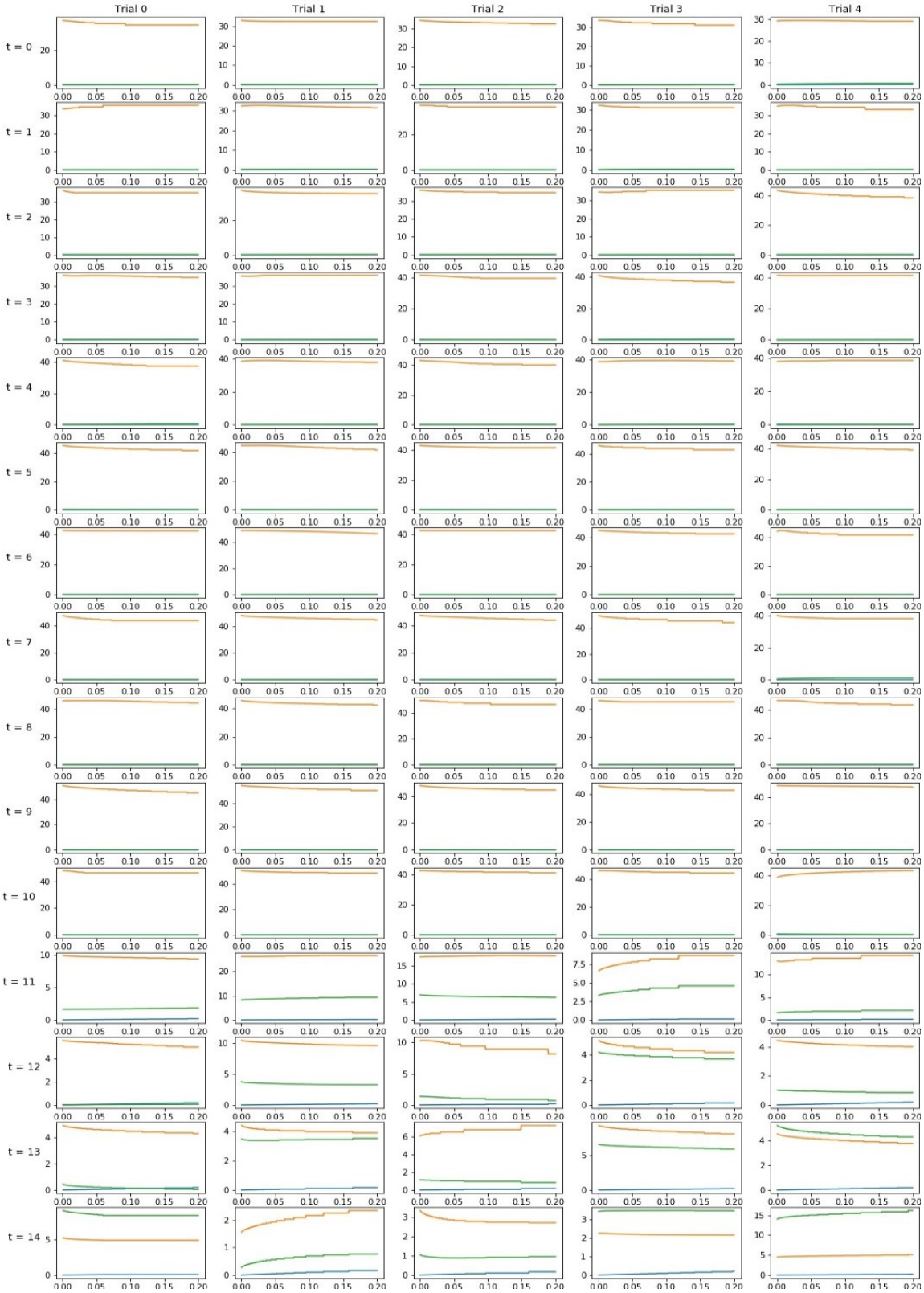


Figure C.1. MSE on observed variables spared for optimization (blue), MSE on observed variables spared for validation (green), and MSE on unobserved variables (orange) over the second 15 time steps with respect to ϵ.

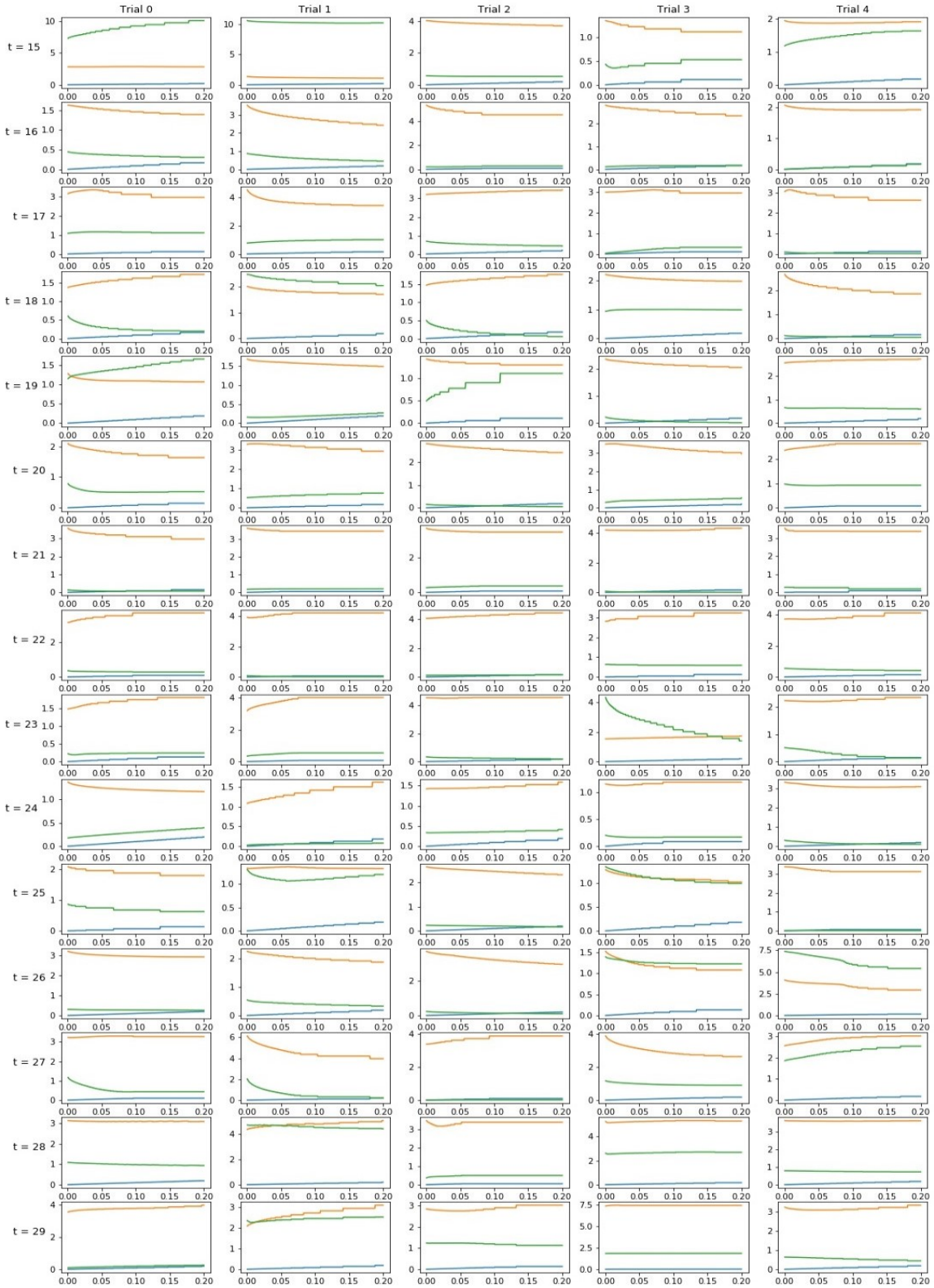


Figure C.2. MSE on observed variables spared for optimization (blue), MSE on observed variables spared for validation (green), and MSE on unobserved variables (MSE) over the second 15 time steps with respect to ϵ. Each row corresponds to a time step, each column corresponds to a trial.

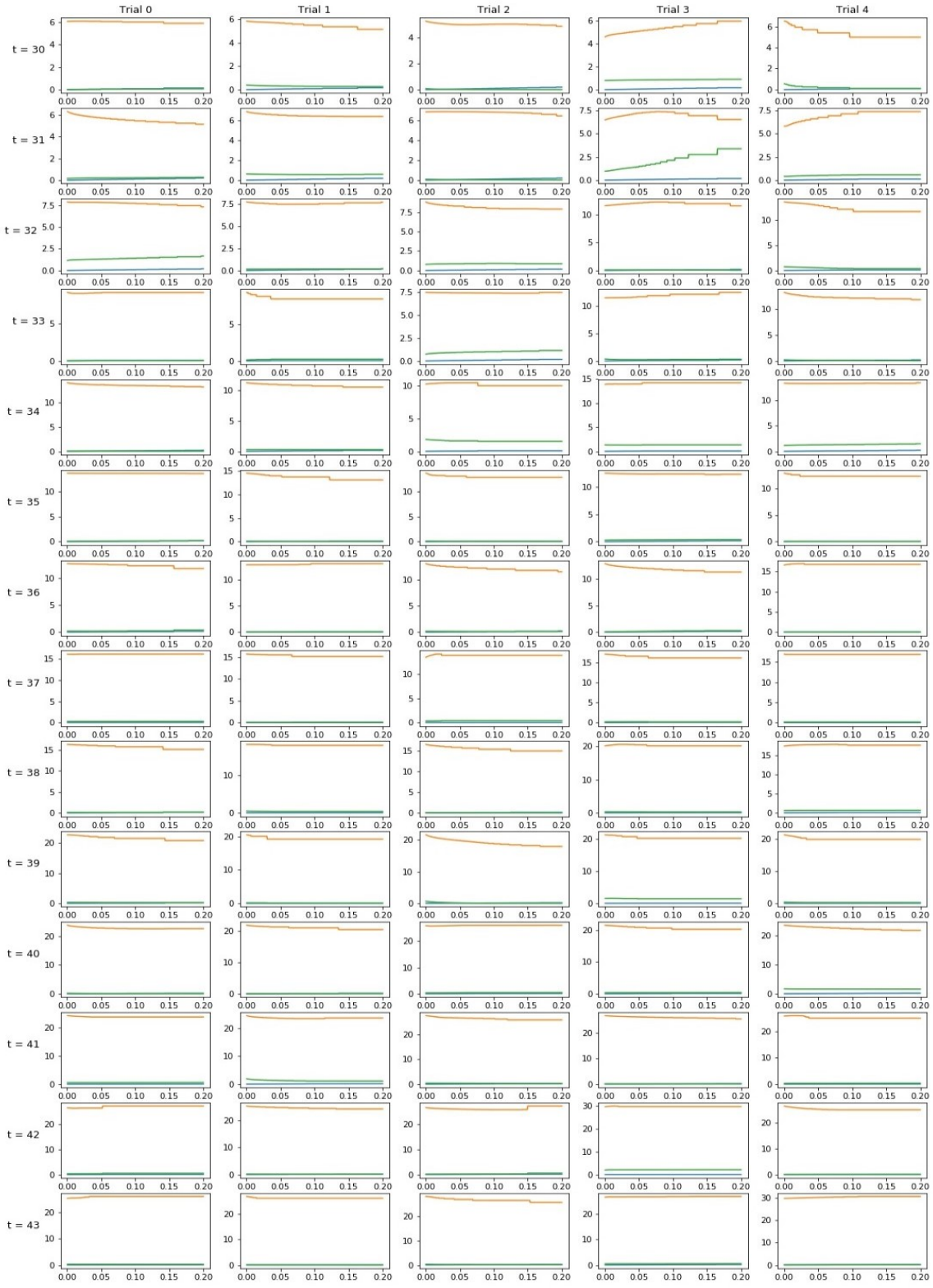


Figure C.3. MSE on observed variables spared for optimization (blue), MSE on observed variables spared for validation (green), and MSE on unobserved variables (MSE) over the last 14 time steps with respect to ϵ. Each row corresponds to a time step, each column corresponds to a trial.

BIBLIOGRAPHY

[1] M. Adya and F. Collopy. How effective are neural networks at forecasting and prediction? a review and evaluation. *Journal of forecasting*, 17(5-6):481–495, 1998.

[2] B. Akar, B. Jiang, S. I. Somo, A. A. Appel, J. C. Larson, K. M. Tichauer, and E. M. Brey. Biomaterials with persistent growth factor gradients in vivo accelerate vascularized tissue formation. *Biomaterials*, 72:61–73, 2015.

[3] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless sensor networks: a survey. *Computer networks*, 38(4):393–422, 2002.

[4] E. Alpaydin. *Introduction to machine learning*. MIT press, 2009.

[5] A. Artel, H. Mehdizadeh, Y.-C. Chiu, E. M. Brey, and A. Cinar. An agent-based model for the investigation of neovascularization within porous scaffolds. *Tissue Engineering Part A*, 17(17-18):2133–2141, 2011.

[6] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[7] A. M. Bailey, B. C. Thorne, and S. M. Peirce. Multi-cell agent-based simulation of the microvasculature to study the dynamics of circulating inflammatory cell trafficking. *The Journal of the Biomedical Engineering Society*, 35(6):916 – 936, 2007.

[8] L. Bao and S. S. Intille. Activity recognition from user-annotated acceleration data. In *International conference on pervasive computing*, pages 1–17. Springer, 2004.

[9] R. H. Bartels, J. C. Beatty, and B. A. Barsky. *An introduction to splines for use in computer graphics and geometric modeling*. Morgan Kaufmann, 1995.

[10] Y. Bengio, R. Ducharme, P. Vincent, and C. Jauvin. A neural probabilistic language model. *Journal of machine learning research*, 3(Feb):1137–1155, 2003.

[11] K. Bentley, H. Gerhardt, and P. A. Bates. Agent-based simulation of notch-mediated tip cell selection in angiogenic sprout initialisation. *Journal of Theoretical Biology*, 250(1):25 – 36, 2008.

[12] P. Bianco and P. G. Robey. Stem cells in tissue engineering. *Nature*, 414(6859): 118–121, 2001.

[13] M. Bilgic. *Cost-Sensitive Information Acquisition in Structured Domains*. PhD thesis, University of Maryland - College Park, August 2010.

[14] M. Bilgic and L. Getoor. Link-based active learning. In *NIPS Workshop on Analyzing Networks and Learning with Graphs*, 2009.

[15] M. Bilgic and L. Getoor. Reflect and correct: A misclassification prediction approach to active inference. *ACM Transactions on Knowledge Discovery from Data*, 3(4):1–32, November 2009.

[16] M. Bilgic and L. Getoor. Active inference for collective classification. In *Twenty-Fourth Conference on Artificial Intelligence (AAAI NECTAR Track)*, pages 1652–1655, 2010.

[17] M. Bilgic and L. Getoor. Value of information lattice: Exploiting probabilistic independence for effective feature subset acquisition. *Journal of Artificial Intelligence Research (JAIR)*, 41:69–95, 2011.

[18] M. Bilgic, L. Mihalkova, and L. Getoor. Active learning for networked data. In *International Conference on Machine Learning*, 2010.

[19] J. Bromley, N. A. Jackson, O. J. Clymer, A. M. Giacomello, and F. V. Jensen. The use of Hugin to develop Bayesian networks as an aid to integrated water resource planning. *Environmental Modelling & Software*, 20(2):231–242, 2005.

[20] D. P. Byrne, D. Lacroix, J. A. Planell, D. J. Kelly, and P. J. Prendergast. Simulation of tissue differentiation in a scaffold as a function of porosity, young's modulus and dissolution rate: Application of mechanobiological models in tissue engineering. *Biomaterials*, 28(36):5544 – 5554, 2007.

[21] G. S. v. d. W. C. H. Anderson, P. J. Burt. Change detection and tracking using pyramid transform techniques. In *Intelligent Robots and Computer Vision IV*, volume 0579. International Society for Optics and Photonics, 1985. doi: 10.1117/12.950785. URL `https://doi.org/10.1117/12.950785`.

[22] E. Castillo, J. Gutiérrez, A. Hadi, and C. Solares. Symbolic propagation and sensitivity analysis in gaussian bayesian networks with application to damage assessment. *Artificial Intelligence in Engineering*, 11(2):173 – 181, 1997. ISSN 0954-1810. doi: http://dx.doi.org/10.1016/S0954-1810(96)00030-1. URL `http://www.sciencedirect.com/science/article/pii/S0954181096000301`.

[23] E. Castillo, J. M. Menéndez, and S. Sánchez-Cambronero. Predicting traffic flow using bayesian networks. *Transportation Research Part B: Methodological*, 42(5):482 – 509, 2008. ISSN 0191-2615. doi: http://dx.doi.org/10.1016/j.trb.2007.10.003. URL `http://www.sciencedirect.com/science/article/pii/S0191261507001300`.

[24] O. Chapelle and Y. Zhang. A dynamic Bayesian network click model for web search ranking. In *Proceedings of the 18th International Conference on World Wide Web*, WWW '09, pages 1–10. ACM, 2009. ISBN 978-1-60558-487-4.

[25] E. Charniak. Bayesian networks without tears. *AI magazine*, 12(4):50–50, 1991.

[26] E. Charniak and R. Goldman. Plan recognition in stories and in life. In *Machine Intelligence and Pattern Recognition*, volume 10, pages 343–351. Elsevier, 1990.

[27] E. Charniak and R. P. Goldman. A semantics for probabilistic quantifier-free first-order languages, with particular application to story understanding. In *IJCAI*, volume 89, pages 1074–1079. Citeseer, 1989.

[28] D. Chen, M. Bilgic, L. Getoor, and D. Jacobs. Dynamic processing allocation in video. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:2174–2187, 2011.

[29] D. Chen, M. Bilgic, L. Getoor, D. Jacobs, L. Mihalkova, and T. Yeh. Active inference for retrieval in camera networks. In *Workshop on Person Oriented Vision*, 2011.

[30] Y.-C. Chiu, M.-H. Cheng, H. Engel, S.-W. Kao, J. C. Larson, S. Gupta, and E. M. Brey. The role of pore size on vascularization and tissue remodeling in {PEG} hydrogels. *Biomaterials*, 32(26):6045 – 6051, 2011. ISSN 0142-9612. doi: http://dx.doi.org/10.1016/j.biomaterials.2011.04.066. URL `http://www.sciencedirect.com/science/article/pii/S0142961211004881`.

[31] Y.-C. Chiu, S. Kocagoz, J. C. Larson, and E. M. Brey. Evaluation of physical and mechanical properties of porous poly (ethylene glycol)-co-(l-lactic acid) hydrogels during degradation. *PLoS One*, 8:4, 2013.

[32] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. In *Conference on Empirical Methods in Natural Language Processing (EMNLP 2014)*, 2014.

[33] D. Cohn, L. Atlas, and R. Ladner. Improving generalization with active learning. *Machine Learning*, 15(2):201–221, 1994.

[34] J. Connor and L. Atlas. Recurrent neural networks and time series prediction. In *IJCNN-91-Seattle International Joint Conference on Neural Networks*, volume 1, pages 301–306. IEEE, 1991.

[35] J. Connor, L. E. Atlas, and D. R. Martin. Recurrent networks and narma modeling. In *Advances in Neural Information Processing Systems*, pages 301–308, 1992.

[36] J. T. Connor, R. D. Martin, and L. E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE transactions on neural networks*, 5(2):240–254, 1994.

[37] I. Dagan and S. P. Engelson. Committee-based sampling for training probabilistic classifiers. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 150–157. The Morgan Kaufmann series in machine learning,(San Francisco, CA, USA), 1995.

[38] T. L. Dean, K. Basye, R. Chekaluk, S. Hyun, M. Lejter, and M. Randazza. Coping with uncertainty in a control system for navigation and exploration. In *AAAI*, pages 1010–1015, 1990.

[39] A. Deshpande, C. Guestrin, S. R. Madden, J. M. Hellerstein, and W. Hong. Model-driven data acquisition in sensor networks. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 588–599. VLDB Endowment, 2004.

[40] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[41] T. Dozat. Incorporating nesterov momentum into adam. 2016.

[42] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[43] E. Elnahrawy and B. Nath. Context-aware sensors. In *Wireless Sensor Networks*, pages 77–93. Springer, 2004.

[44] J. Forbes, T. Huang, K. Kanazawa, and S. Russell. The BATmobile: Towards a Bayesian automated taxi. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, IJCAI-95*, volume 2, 1995.

[45] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[46] Z. Ghahramani. Learning dynamic bayesian networks. In *Adaptive processing of sequences and data structures*, pages 168–197. Springer Berlin Heidelberg, 1998.

[47] C. L. Giles, G. M. Kuhn, and R. J. Williams. Dynamic recurrent neural networks: Theory and applications. *IEEE Transactions on Neural Networks*, 5(2): 153–156, 1994.

[48] R. P. Goldman. A probabilistic approach to language understanding. 1990.

[49] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*. MIT press, 2016.

[50] W. L. Gorr, D. Nagin, and J. Szczypula. Comparative study of artificial neural network and statistical models for predicting student grade point averages. *International Journal of Forecasting*, 10(1):17 – 34, 1994. ISSN 0169-2070. doi: https://doi.org/10.1016/0169-2070(94)90046-9. URL `http://www.sciencedirect.com/science/article/pii/0169207094900469`.

[51] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[52] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 18 (5-6):602–610, 2005.

[53] A. Graves, S. Fernández, and J. Schmidhuber. Bidirectional LSTM networks for improved phoneme classification and recognition. In *International Conference on Artificial Neural Networks*, pages 799–804. Springer, 2005.

[54] A. Graves, M. Liwicki, S. Fernández, R. Bertolami, H. Bunke, and J. Schmidhuber. A novel connectionist system for unconstrained handwriting recognition. *IEEE transactions on pattern analysis and machine intelligence*, 31(5):855–868, 2009.

[55] O. Hansson and A. Mayer. Heuristic search as evidential reasoning. In *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, pages 152–161. Association for Uncertainty in Artificial Intelligence, Mountain View, CA, 1989.

[56] S. Haykin. *Neural networks*, volume 2. Prentice hall New York, 1994.

[57] D. Heckerman. Probabilistic similarity networks (technical report stan-cs-1316). *Palo Alto, CA: Stanford University, Departments of Computer Science and Medicine*, 1990.

[58] K.-L. Ho, Y.-Y. Hsu, and C.-C. Yang. Short term load forecasting using a multilayer neural network with an adaptive learning algorithm. *IEEE Transactions on Power Systems*, 7(1):141–149, 1992.

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

[60] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[61] M. C. Horsch. *Dynamic Bayesian networks*. PhD thesis, University of British Columbia, 1990. URL `https://open.library.ubc.ca/cIRcle/collections/831/items/1.0051146`.

[62] D. Husmeier. Sensitivity and specificity of inferring genetic regulatory interactions from microarray experiments with dynamic Bayesian networks. *Bioinformatics*, 19(17):2271–2282, 2003.

[63] B. Jiang, B. Akar, T. Waller, J. Larson, A. Appel, and E. Brey. Design of a composite biomaterial system for tissue engineering applications. *Acta Biomaterialia*, 2013.

[64] B. Jiang, T. M. Waller, J. C. Larson, A. A. Appel, and E. M. Brey. Fibrin-loaded porous poly(ethylene glycol) hydrogels as scaffold materials for vascularized tissue formation. *Tissue engineering. Part A*, 19:224–234, 2013.

[65] R. E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Fluids Engineering*, 82(1):35–45, 1960.

[66] B. Karlik and A. V. Olgac. Performance analysis of various activation functions in generalized mlp architectures of neural networks. *International Journal of Artificial Intelligence and Expert Systems*, 1(4):111–122, 2011.

[67] M. Kass, A. Witkin, and D. Terzopoulos. Snakes: Active contour models. *International journal of computer vision*, 1(4):321–331, 1988.

[68] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *Proceedings of International Conference on Learning Representations*, pages 1 – 13, 2015.

[69] A. Klaser, M. Marszalek, and C. Schmid. A Spatio-Temporal Descriptor Based on 3D-Gradients. In M. Everingham, C. Needham, and R. Fraile, editors, *BMVC 2008 - 19th British Machine Vision Conference*, pages 275:1–10, Leeds, United Kingdom, Sept. 2008. British Machine Vision Association. URL `https://hal.inria.fr/inria-00514853`.

[70] D. Koller and N. Friedman. *Probabilistic Graphical Models: Principles and Techniques*. The MIT Press, 2009. ISBN 0262013193, 9780262013192.

[71] C. Komurlu, J. Shao, and M. Bilgic. Dynamic bayesian network modeling of vascularization in engineered tissues. In *Proceedings of the Eleventh UAI Bayesian Modeling Applications Workshop*, 2014. URL `http://www.cs.iit.edu/~ml/pdfs/komurlu-bmaw14.pdf`.

[72] A. Krause and C. Guestrin. Optimal nonmyopic value of information in graphical models - efficient algorithms and theoretical limits. In *International Joint Conference on Artificial Intelligence*, pages 1339–1345, 2005.

[73] A. Krause and C. Guestrin. Near-optimal nonmyopic value of information in graphical models. In *Annual Conference on Uncertainty in Artificial Intelligence*, 2005.

[74] A. Krause and C. Guestrin. Optimal value of information in graphical models. *Journal of Artificial Intelligence Research*, 35:557–591, 2009.

[75] C.-M. Kuan and T. Liu. Forecasting exchange rates using feedforward and recurrent neural networks. *Journal of applied econometrics*, 10(4):347–364, 1995.

[76] J. R. Kwapisz, G. M. Weiss, and S. A. Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.

[77] I. Laptev, M. Marszalek, C. Schmid, and B. Rozenfeld. Learning Realistic Human Actions from Movies. In *CVPR 2008 - IEEE Conference on Computer Vision & Pattern Recognition*, pages 1–8, Anchorage, United States, June 2008. IEEE Computer Society. doi: 10.1109/CVPR.2008.4587756. URL `https://hal.inria.fr/inria-00548659`.

[78] T. S. Levitt, J. M. Agosta, and T. O. Binford. Model-based influence diagrams for machine vision. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, pages 371–388. North-Holland Publishing Co., 1990.

[79] D. D. Lewis and W. A. Gale. A sequential algorithm for training text classifiers. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 3–12, 1994.

[80] W.-J. Li, R. Tuli, C. Okafor, A. Derfoul, K. G. Danielson, D. J. Hall, and R. S. Tuan. A three-dimensional nanofibrous scaffold for cartilage tissue engineering using human mesenchymal stem cells. *Biomaterials*, 26(6):599–609, 2005.

[81] A. J. Lipton, H. Fujiyoshi, and R. S. Patil. Moving target classification and tracking from real-time video. In *Proceedings Fourth IEEE Workshop on Applications of Computer Vision. WACV'98 (Cat. No. 98EX201)*, pages 8–14. IEEE, 1998.

[82] A. M. Logar. *Recurrent Neural Networks and Time Series Prediction*. PhD thesis, Texas Tech University, December 1992. URL `https://ttu-ir.tdl.org/bitstream/handle/2346/60697/31295007106999.pdf?sequence=1f`.

[83] P. Malhotra, L. Vig, G. Shroff, and P. Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, page 89. Presses universitaires de Louvain, 2015.

[84] J. Mao, W. Giannobile, J. Helms, S. Hollister, P. Krebsbach, M. Longaker, and S. Shi. Craniofacial tissue engineering by stem cells. *Journal of dental research*, 85(11):966–979, 2006.

[85] D. Margaritis. *Learning Bayesian Network Model Structure from Data*. PhD thesis, Citeseer, 2003.

[86] A. McCallum, K. Nigam, et al. A comparison of event models for naive bayes text classification. In *AAAI-98 workshop on learning for text categorization*, volume 752, pages 41–48. Citeseer, 1998.

[87] H. Mehdizadeh, A. Artel, E. M. Brey, and A. Cinar. Multi-agent systems for biomedical simulation: modeling vascularization of porous scaffolds. In *Agents in Principle, Agents in Practice*, pages 113–128. Springer Berlin Heidelberg, 2011.

[88] H. Mehdizadeh, S. Sumo, E. S. Bayrak, E. M. Brey, and A. Cinar. Three-dimensional modeling of angiogenesis in porous biomaterial scaffolds. *Biomaterials*, 34(12):2875 – 2887, 2013.

[89] Z. Min and S. D. Conzen. A new dynamic Bayesian network (DBN) approach for identifying gene regulatory networks from time course microarray data. *Bioinformatics*, 21(1):71–79, 2005.

[90] K. P. Murphy. *Dynamic Bayesian Networks: Representation, Inference and Learning*. PhD thesis, University of California, Berkeley, 1994.

[91] K. P. Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012. ISBN 9780262018029. doi: 10.1007/SpringerReference_35834. URL `http://link.springer.com/chapter/10.1007/978-94-011-3532-0{_}2`.

[92] S. K. Nayar and T. Poggio. *Early visual learning*, volume 11. Oxford University Press New York, NY, 1996.

[93] J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

[94] Y. Nesterov. A method for unconstrained convex minimization problem with the rate of convergence o (1/kˆ 2). In *Doklady AN USSR*, volume 269, pages 543–547, 1983.

[95] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[96] V. Pavlovic, J. M. Rehg, T.-J. Cham, and K. P. Murphy. A dynamic Bayesian network approach to figure tracking using learned dynamic models. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 94–101 vol.1, 1999.

[97] J. Pearl. *Probabilistic Reasoning in Intelligent Systems*. Morgan Kaufmann, San Francisco, 1988.

[98] B. A. Pearlmutter. Learning state space trajectories in recurrent neural networks. *Neural Computation*, 1(2):263–269, 1989.

[99] B.-E. Perrin, L. Ralaivola, A. Mazurie, S. Bottani, J. Mallet, and F. d'Alche Buc. Gene networks inference using dynamic Bayesian networks. *Bioinformatics*, 19(suppl 2):ii138–ii148, 2003.

[100] J. R. Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[101] M. E. Ramirez-Loaiza. *Anytime Active Learning*. PhD thesis, Illinois Institute of Technology, May 2016. URL `http://www.cs.iit.edu/~ml/pdfs/ramirez-phdthesis16.pdf`.

[102] M. E. Ramirez-Loaiza, A. Culotta, and M. Bilgic. Towards anytime active learning: Interrupting experts to reduce annotation costs. In *IDEA Workshop at ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, 2013.

[103] M. E. Ramirez-Loaiza, A. Culotta, and M. Bilgic. Anytime Active Learning. In *AAAI Conference on Artificial Intelligence*, 2014. URL `http://www.cs.iit.edu/~ml/pdfs/ramirez-aaai14.pdf`.

[104] M. E. Ramirez-Loaiza, M. Sharma, G. Kumar, and M. Bilgic. Active learning: an empirical study of common baselines. *Data Mining and Knowledge Discovery*, 31(2):287–313, 2017. ISSN 1573-756X. doi: 10.1007/s10618-016-0469-7. URL `http://www.cs.iit.edu/~ml/pdfs/ramirez-dmkd17.pdf`.

[105] M. Ranzato, A. Szlam, J. Bruna, M. Mathieu, R. Collobert, and S. Chopra. Video (language) modeling: a baseline for generative models of natural videos. *arXiv preprint arXiv:1412.6604*, 2014.

[106] C. E. Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

[107] M. Rattigan, M. Maier, and D. Jensen. Exploiting network structure for active inference in collective classification. In *ICDM Workshop on Mining Graphs and Complex Structures*, pages 429–434, 2007.

[108] T. Robinson. An application of recurrent nets to phone probability estimation. *IEEE transactions on Neural Networks*, 5(2), 1994.

[109] T. Robinson, M. Hochberg, and S. Renals. The use of recurrent neural networks in continuous speech recognition. In *Automatic speech and speaker recognition*, pages 233–258. Springer, 1996.

[110] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[111] N. Roy and A. McCallum. Toward optimal active learning through sampling estimation of error reduction. In *International Conference on Machine Learning*, pages 441–448, 2001.

[112] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[113] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[114] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 1986.

[115] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition edition, 2010.

[116] H. Sak, A. Senior, and F. Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[117] H. Sak, A. Senior, and F. Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Fifteenth annual conference of the international speech communication association*, 2014.

[118] W. S. Sarle. Stopped training and other remedies for overfitting. *Computing science and statistics*, pages 352–360, 1996.

[119] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective classification in network data. *AI Magazine*, 29(3):93–106, 2008.

[120] B. Settles. *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool, 2012.

[121] M. Sharma. *Active Learning with Rich Feedback*. PhD thesis, Illinois Institute of Technology, July 2017. URL `http://www.cs.iit.edu/~ml/pdfs/sharma-phdthesis17.pdf`.

[122] M. Sharma and M. Bilgic. Most-surely vs. least-surely uncertain. In *IEEE 13th International Conference on Data Mining (ICDM)*, 2013.

[123] M. Sharma and M. Bilgic. Towards learning with feature-based explanations for document classification. In *IJCAI Workshop on BeyondLabeler - Human is More than a Labeler*, 2016. URL `http://www.cs.iit.edu/~ml/pdfs/sharma-ijcai-bl16.pdf`.

[124] M. Sharma and M. Bilgic. Evidence-based uncertainty sampling for active learning. *Data Mining and Knowledge Discovery*, 31(1):164–202, 2017.

[125] M. Sharma and M. Bilgic. Learning with rationales for document classification. *Machine Learning*, 2017. doi: 10.1007/s10994-017-5671-3. URL `http://www.cs.iit.edu/~ml/pdfs/sharma-ml17.pdf`.

[126] M. Sharma, D. Zhuang, and M. Bilgic. Active learning with rationales for text classification. In *North American Chapter of the Association for Computational Linguistics – Human Language Technologies*, 2015. URL `http://www.cs.iit.edu/~ml/pdfs/sharma-naaclhlt15.pdf`.

[127] M. Sharma, K. Das, M. Bilgic, B. Matthews, D. Nielsen, and N. Oza. Active learning with rationales for identifying operationally significant anomalies in aviation. In *Proceedings of European Conference on Machine Learning and Principles and Practice of Knowledge Discovery (ECML-PKDD)*, 2016. URL `http://www.cs.iit.edu/~ml/pdfs/sharma-ecmlpkdd16.pdf`.

[128] P. Sibi, S. A. Jones, and P. Siddarth. Analysis of different activation functions using back propagation neural networks. *Journal of Theoretical and Applied Information Technology*, 47(3):1264–1268, 2013.

[129] S. Slijepcevic and M. Potkonjak. Power efficient organization of wireless sensor networks. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 2, pages 472–476 vol.2, 2001. doi: 10.1109/ICC.2001.936985.

[130] S. I. Somo, B. Akar, E. S. Bayrak, J. C. Larson, A. A. Appel, H. Mehdizadeh, A. Cinar, and E. M. Brey. Pore interconnectivity influences growth factor mediated vascularization in sphere templated hydrogels. *Tissue Engineering*, (ja), 2015.

[131] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[132] D. J. Spiegelhalter, R. C. Franklin, and K. Bull. Assessment, criticism and improvement of imprecise subjective probabilities for a medical expert system. In *Proceedings of the Fifth Annual Conference on Uncertainty in Artificial Intelligence*, pages 285–294. North-Holland Publishing Co., 1990.

[133] N. Srivastava, E. Mansimov, and R. Salakhudinov. Unsupervised learning of video representations using lstms. In *International conference on machine learning*, pages 843–852, 2015.

[134] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[135] S. Tan and M. L. Mayrovouniotis. Reducing data dimensionality through optimizing neural network inputs. *AIChE Journal*, 41(6):1471–1480, 1995.

[136] D. Terzopoulos and R. Szeliski. Tracking with kalman snakes. *Active vision*, 20:3–20, 1992.

[137] T. Tommasini, A. Fusiello, V. Roberto, and E. Trucco. Robust feature tracking in underwater video sequences. In *IEEE Oceanic Engineering Society. OCEANS'98. Conference Proceedings (Cat. No. 98CH36259)*, volume 1, pages 46–50. IEEE, 1998.

[138] E. Trucco and K. Plakas. Video tracking: a concise survey. *IEEE Journal of Oceanic Engineering*, 31(2):520–529, 2006.

[139] E. Trucco and A. Verri. *Introductory techniques for 3-D computer vision*, volume 201. Prentice Hall Englewood Cliffs, 1998.

[140] L. Uusitalo. Advantages and challenges of Bayesian networks in environmental modelling. *Ecological modelling*, 3(203):312–318, 2007.

[141] L. Wang and A. Deshpande. Predictive modeling-based data collection in wireless sensor networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4913 LNCS:34–51, 2008. ISSN 03029743. doi: 10.1007/978-3-540-77690-1{_}3.

[142] P. J. Werbos. Generalization of backpropagation with application to a recurrent gas market model. *Neural networks*, 1(4):339–356, 1988.

[143] P. J. Werbos et al. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560, 1990.

[144] M. R. Wick and W. B. Thompson. Reconstructive expert system explanation. *Artificial Intelligence*, 54(1-2):33–70, Mar. 1992.

[145] S. Xingjian, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015.

[146] S. Yang, K.-F. Leong, Z. Du, and C.-K. Chua. The design of scaffolds for use in tissue engineering. part I. traditional factors. *Tissue Engineering*, 7:679–689, 2004.

[147] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[148] G. Zhang, B. E. Patuwo, and M. Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1): 35–62, 1998.

[149] L. Zhang, J. Zhu, and T. Yao. An evaluation of statistical spam filtering techniques. *ACM Transactions on Asian Language Information Processing (TALIP)*, 3(4):243–269, 2004.

[150] H. Zhou and H. Hu. Human motion tracking for rehabilitationa survey. *Biomedical Signal Processing and Control*, 3(1):1 – 18, 2008. ISSN 1746-8094. doi: https://doi.org/10.1016/j.bspc.2007.09.001. URL `http://www.sciencedirect.com/science/article/pii/S1746809407000778`.

[151] G. Zweig and S. Russell. Speech recognition with dynamic Bayesian networks. In *AAAI-98 Proceedings*, 1998.