DEEP SENSING: ACTIVE SENSING USING MULTIDIRECTIONAL RECURRENT NEURAL NETWORKS

Anonymous authors
Paper under double-blind review

ABSTRACT

For every prediction we might wish to make, we must decide what to observe (what source of information) and when to observe it. Because making observations is costly, this decision must trade off the value of information against the cost of observation. Making observations (sensing) should be an active choice. To solve the problem of active sensing we develop a novel deep learning architecture: Deep Sensing. At training time, Deep Sensing learns how to issue predictions at various cost-performance points. To do this, it creates multiple representations at various performance levels associated with different measurement rates (costs). This requires learning how to estimate the value of real measurements vs. inferred measurements, which in turn requires learning how to infer missing (unobserved) measurements. To infer missing measurements, we develop a Multi-directional Recurrent Neural Network (M-RNN). An M-RNN differs from a bi-directional RNN in that it sequentially operates across streams in addition to within streams, and because the timing of inputs into the hidden layers is both lagged and advanced. At runtime, the operator prescribes a performance level or a cost constraint, and Deep Sensing determines what measurements to take and what to infer from those measurements, and then issues predictions. To demonstrate the power of our method, we apply it to two real-world medical datasets with significantly improved performance.

1 INTRODUCTION

Making observations is costly. Hence, for every prediction we might wish to make, we must decide what to observe – i.e., what source of information to consult/use – and when to observe it. This (joint) decision involves a trade-off between the value of the information that will/might be obtained from the observation and the cost of making the observation. There is little reason to make an observation if the result of that observation can already be confidently estimated on the basis of what is already known or if the result would be of little value in any case; it would be much better to conserve the resources to make a different observation at a different time. Thus making observations (sensing) should be an active choice (Yu et al. [2009]). The problem of active sensing has many applications, from healthcare (the example we use here to illustrate our method) to neuroscience to robotics to wireless communications.

The central point of our approach is that we need to estimate the value of information. This must be learned at training time. We learn the estimated value for a specified set of measurements by first predicting on the basis of the information we have, then deleting the specified set of measurements, inferring what we have deleted on the basis of the data that remains, making a new prediction on the basis of the inferred measurements and the remaining data, and comparing the two predictions. (Part of our architecture is designed specifically for these tasks.)

To infer missing data, we develop a novel architecture called a Multi-directional Recurrent Neural Network (M-RNN). Like a bi-directional RNN (Bi-RNN) (Graves & Schmidhuber [2005]), an M-RNN operates forward and backward in each data stream – in the intra-stream directions – and across streams – in the inter-stream directions. Unlike a Bi-RNN, the timing of inputs into the hidden layers of an M-RNN is lagged in the forward direction and advanced in the backward direction. (To the best of our knowledge, our architecture is the first that operates in this way). Our M-RNN executes both interpolation (intra-stream) and imputation (inter-stream) to infer missing data.
Because we need to trade off performance against cost, our neural network must learn – at training time – how to issue predictions at various cost-performance points. To do this, it creates multiple representations (neural network parameters) at various performance levels associated with different measurement rates (costs). Each representation is learned on the basis of a particular set of missing data; these sets are constructed recursively and adaptively.

An important aspect of our solution to the active sensing problem is that there are important differences between the operation at training time and runtime. At training time we can use data at the current time to infer missing data at an earlier time; i.e. we can operate non-causally. We cannot do so – and do not do so – at runtime. However, after a new sample is received at runtime, we can and do go back to improve the previous inferences (interpolations and imputations) which will in turn improve both current and future predictions.

To demonstrate the power of our method, we apply it to two real-world medical datasets. We show that our method yields significantly greater predictive power (measured as the Area Under the ROC Curve (AUC)) per unit cost in comparison to other state-of-the-art methods. Because our inference methods are of interest in themselves, we compare the root mean squared error (RMSE) and curve (AUC) per unit cost in comparison to other state-of-the-art methods. Because our inference

2 Deep Sensing

2.1 Data

The training set consists of \( N \) arrays of data. It is convenient to use the language of healthcare and to speak of the array \( n \) as the information of patient \( n \); so that there are \( N \) patients in the training set. For each patient \( n \), we have a multivariate time-series data stream of length \( T \) (the length \( T \) and the other components may depend on the patient \( n \) but for the moment we suppress the dependence on \( n \)) that consists of three components: measurements \( X \), labels \( Y \) and time stamps \( S \).

Because measurements are not necessarily made at regular intervals, we distinguish between time stamps and actual times. The time stamp \( t = 1, 2, \ldots \) simply indexes the sequence of times at which measurements were taken; \( s_i \) is the actual time at which the measurements \( x_i \) were taken and the label \( y_i \) was realized. For convenience we normalize so that \( s_1 = 0 \); we assume actual times are

The label \( y_t \) represents the outcome realized for patient \( n \) at time stamp \( t \) (actual time \( s_t \)). Labels may be discrete or continuous. In the former case we are considering a classification problem (e.g. prediction of an event, such as discharge, clinical deterioration, death); in the latter case, we are considering a regression problem (e.g. prediction of value or family of values). If we are interested explicitly in the estimation of missing data for its own sake, then \( y_t \) would represent the actual observed data at time stamp \( t \). \( Y \) is the vector of outcomes for this patient. We normalize so that labels and predictions lie in \([0, 1]\).

There are \( D \) streams of measurements; each measurement is a real number, but not all measurements may be observed at each time stamp. Hence we view the set of possible measurements at time stamp \( t \) as \( \mathbb{R}_* = \mathbb{R} \cup \{*\} \). We interpret \( x_i^d = * \) to mean that the stream \( d \) was not measured at time stamp \( t \); otherwise \( x_i^d \in \mathbb{R} \) is the measurement of stream \( d \) at time stamp \( t \). \( X \) is the array of measurements of all streams at all time stamps for the patient under consideration.

It is convenient to introduce some notation to keep track of what is measured/not measured (observed/not observed). For each time stamp \( t \) and stream \( d \), write \( m_{t}^{d} = 0 \) if \( x_{t}^{d} = * \) (not measured) and \( m_{t}^{d} = 1 \) if \( x_{t}^{d} \in \mathbb{R} \) (measured). Let \( \delta_{t}^{d} \) be the actual amount of time that has elapsed since the stream \( d \) was measured last. \( \delta_{t}^{d} \) can be defined recursively as follows:

\[
\delta_{t}^{d} = \begin{cases} 
    s_t - s_{t-1} + \delta_{t-1}^{d} & \text{if } t > 1, m_{t-1}^{d} = 0, \\
    s_t - s_{t-1} & \text{if } t > 1, m_{t-1}^{d} = 1
\end{cases}
\]

where \( \delta_{1}^{d} = 0 \). Write \( \delta_{t} \) for the vector of elapsed times at time stamp \( t \) and \( \Delta = \{\delta_{1}, \delta_{2}, \ldots, \delta_{T}\} \).
The information available for patient $n$ is the triple $(X_n, Y_n, S_n)$. The entire training set therefore is the sets of triples $D = \{(X_n, Y_n, S_n)\}_{n=1}^N$. We use functional notation to identify information about each patient, so $x_t^d(n)$ is the measurement of stream $d$ at time stamp $t$ for patient $n$, etc.

### 2.2 Active Sensing

At time stamp $T$, we have an array of measurements (which may or may not include the current label $y_T$); we must decide the next time $s_{T+1}$ at which to take new measurements and what measurements to conduct at that time. We measure the information provided by new samples by the effect on the label $y_{T+1}$, so we define the predictive loss from not sampling as the increase in uncertainty of our prediction of $y_{T+1}$ and the predictive gain as the decrease in uncertainty of our prediction of $y_{T+1}$. Our approach is to find the first actual time $\tau$ at which the (estimated) predictive gain provided by new samples exceeds the cost of sampling (keeping in mind that the cost of sampling may be different for different streams), and to make the set of measurements at time $\tau$ that maximizes the (estimated) predictive gain minus the cost.

The actual error in sampling stream $d$ at $s_{T+1} = \tau$ is the difference between the estimated values and the actual measurement $e_{T+1}^d = |\hat{x}_{T+1}^d - x_{T+1}^d|$. We don’t know the actual error so we must construct an estimated error $\hat{e}_{T+1}^d$. The confidence intervals in the measurement of $x_{T+1}^d$ are of the form $CI_d = (\hat{x}_{T+1}^d - \lambda\hat{e}_{T+1}^d, \hat{x}_{T+1}^d + \lambda\hat{e}_{T+1}^d)$; e.g. $\lambda = 1.96$ for the 95% confidence level with normality assumption. Note that the confidence intervals depend only on the estimates and not the true values (which are of course unknown). Each vector of estimates $(\hat{x}_{T+1}^d)$ of measurements, together with previous data (measured and inferred), can be used to produce a prediction $\hat{y}_{T+1}^d$ (see below). The confidence intervals in the measurement of $\hat{x}_{T+1}^d$ translate immediately into lower and upper confidence estimates $\hat{y}_{T+1}^d, \hat{y}_{T+1}^{d,u}$ (respectively) for the label prediction:

$$\hat{y}_{T+1}^{d,l} = \min_{\hat{x}_{T+1}^d \in CI_{T+1}} \hat{y}^d(\hat{x}_{T+1}^d, X_T, S_T) \quad \hat{y}_{T+1}^{d,u} = \max_{\hat{x}_{T+1}^d \in CI_{T+1}} \hat{y}^d(\hat{x}_{T+1}^d, X_T, S_T)$$

where $X_T$ and $S_T$ are previous measurements and measurement times until time stamp $T$. The (estimated) predictive gain in stream $d$ is therefore the difference $\hat{y}_{T+1}^d - \hat{y}_{T+1}^{d,u}$.

Formally, we set $s_{T+1}$ to be the first actual time at which there is some set of measurements for which the (estimated) predictive gain exceeds the cost of sampling. At time $s_{T+1}$, we seek to find the subset $C_{T+1} \subset M$ of measurements that maximizes predictive gain minus cost. If we write predictive gain minus cost as $F(C_{T+1}, X_T, S_T)$, then the formal optimization problem is:

$$C_{T+1} = \arg \max_{C_{T+1} \subset M} F(C_{T+1}, X_T, S_T)$$

$$= \arg \max_{C_{T+1} \subset M} \sum_{d \in C_{T+1}} (\hat{y}_{T+1}^{d,u}(X_T, S_T) - \hat{y}_{T+1}^{d,l}(X_T, S_T) - c^d)$$

where $c^d$ is the cost of sampling from stream $d$.

### Predicting Labels:

Given data (measured and inferred) until any time stamp $T'$, we generate a prediction $\hat{y}_{T'}$. The prediction rule can be learned from training data by any of various machine learning algorithms; we use a standard GRU-based RNN (Chung et al. (2014)). (See Prediction in Section 4).

### Estimating the Values of New Measurements:

We view the problem of estimating new measurements as a special case of estimating missing measurements, so we begin by discussing our novel methods for this problem.

Fix data $D$ through time stamp $T$. Assume that $x_t^d = *$. There are two standard methods to form an estimate $\hat{x}_t^d$: interpolation and imputation. Interpolation uses only the measurements $x_t^d$ of the fixed data stream $d$ for other time stamps $t'$ (perhaps both before and after $t$). Interpolation ignores the correlation with other data streams. Imputation uses only the measurements $x_t^d$ at the fixed time $t$ for other data streams $d'$. Imputation ignores the correlation with other times.

In principle, we could try to form the estimate $\hat{x}_t^d$ by using all the information in $D$. However, this would require learning a vast number of parameters and hence a vast number of training instances, so this is impractical. Instead, we propose an efficient hierarchical learning framework using a novel RNN architecture that allows us to capture the correlations both within streams and across streams.
2.3 ILLUSTRATION

Deep Sensing consists of three main components: an Interpolation & Imputation block, a Prediction block, and an Active Sensing block. The operations of each block are illustrated in Fig. 1. In the above Figure, we consider two data streams: Blood Pressure and Heart Rate. The Interpolation & Imputation block estimates missing measurements using the observed measurements until the current time. The Prediction block uses the observed and estimated missing measurements to make a (mortality) prediction. At the next time, the Active Sensing block selects a stream (in this case, Blood Pressure) from which to take a measurement (on the basis that the information provided by this new measurement exceeds the cost of sampling). In this illustration the other stream (Heart Rate) is not measured at this next time, but rather is estimated in the Interpolation & Imputation block.

3 RELATED WORKS

Previous works related to Deep Sensing fall into four areas: Active sensing, Inference of missing values, Bayesian optimization, and RNN methods for timely prediction with missing values.

Active sensing: As discussed in the Introduction, the focus of active sensing (Yu et al. (2009); Alaa & Van Der Schaar (2016)) and of screening policies (Ahuja et al. (2017)) is to determine what and when to measure; this is an important question whenever acquiring measurements is costly. Yu et al. (2009) studies the problem of active sensing using a Bayesian approach with Gaussian processes. This work models the $D$ data streams as Gaussian processes, so the number of parameters is of order $D^2$ and estimation accuracy decreases dramatically as $D$ grows. Moreover, this work creates only a single representation for the entire data set and hence does not effectively trade off predictive gain against measurement cost, and cannot deal with a setting in which there are different costs to sample different variables. Alaa & Van Der Schaar (2016) addresses the problem of active sensing for a single data stream. That work assumes a specific stochastic process to learn the optimal time to sample the next measurement, given the characteristics of the specific stochastic process. Because this work treats only a single data stream and imposes a particular model of the data, this approach cannot be applied to a general data stream and is ineffective in active sensing across multiple data streams. Ahuja et al. (2017) proposes a methodology for personalized screening in the medical domain but the procedure for learning representations is independent of the screening policy.

Missing value inference: There are two standard methods to deal with missing information in time-series data streams: interpolation and imputation. Interpolation methods (Kreindler & Lumsden (2012); Mondal & Percival (2010)) attempt to capture the temporal relationships within each data stream but not the relationships across streams. Imputation methods (White et al. (2011); Rehfeld et al. (2011); García-Laencina et al. (2010)) attempt to capture the synchronous relationships across data streams but not the temporal relationships within streams. (Most of this work assumes a specific

1The problem of active sensing should not be confused with the problem of active learning (e.g. MacKay (1992); Seung et al. (1992)) which focuses on the costly acquisition of labels – not measurements.
model of the data, rather than learning a representation from the data, as Deep Sensing does.) We are not aware of any previous work that attempts to capture both the relationships within stream and the relationships across streams.

Bayesian optimization: The problem of costly measurements has been studied in other areas as well. Bayesian optimization (Pelikan et al. (1999); Snoek et al. (2012)) uses a Gaussian process regression (GPR) to approximate the loss function for a given optimization problem (Seo et al. (2000)). This approximation is then used to sequentially evaluate the true loss function at points where the expected decrease in loss is the greatest. When function evaluations are computationally costly (e.g. for hyper-parameter optimization in complex problems), this approach is a way of identifying good minima given constraints on time.

There are significant differences between Bayesian optimization and Deep Sensing. Firstly, in Bayesian optimization, “cost” is usually taken to be computation time, and optimization is performed subject to a cost constraint – a maximum number of permissible evaluations. In traditional settings, this cost is essentially treated as constant, and not explicitly considered in selecting points during the optimization procedure. Deep Sensing, on the other hand, trades off cost against (predictive) gain. Secondly, in the active sensing setting we consider, measurements can be taken only forwards in time; in the setting of Bayesian optimization, no restrictions are placed on the location of function evaluations. Active sensing thus captures the problems in the healthcare setting, in which causal predictions are needed to inform the actions of practitioners in a timely fashion. Finally, Bayesian optimization uses GPR to approximate loss functions, which places limitations on the types of functions which it can mimic. Because neural networks are “universal approximators” (Hornik et al. (1989)), the RNNs used in Deep Sensing allow it to model a richer set of functions (and give rise to more complicated and interesting dynamics).

RNN methods with missing values: RNNs have been used successfully for prediction on the basis of time-series data with missing data and irregular sampling. The approach of Gingras & Bengio (1996) is to first replace all the missing information with a mean value and use the feedback loop from the hidden states to update the imputed value while learning the classification problem using a standard RNN. Tresp & Briegel (1998) uses the Expectation-Maximization (EM) algorithm to impute the missing values and uses the reconstructed data streams as inputs to a standard RNN for prediction. As with standard imputation methods, the imputation depends only on the synchronous relationships across data streams and not on the temporal relationships within streams. Parveen & Green (2002) use a linear model to estimate missing values from the latest measurement and the hidden state of each stream. As with standard interpolation methods, the estimate depends only on the temporal relationships within each stream and not on the relationships across streams.

More recent works address both missing values and irregularly sampled time-series data streams (Choi et al. (2015); Lipton et al. (2016); Che et al. (2016); Kim et al. (2017)). These papers use the sampling times to capture the informative missingness and time interval information to deal with irregular sampling. They do this by concatenating the measurements, sampling information and time intervals and using the concatenation as the input of an RNN. These papers differ in the replacements they use for missing values. Choi et al. (2015); Lipton et al. (2016); Kim et al. (2017) replace the missing values with 0, mean values or latest measurements – all of which are independent of either the intra-stream or inter-stream relationships or both. Therefore, those methods cannot be extended to our active sensing algorithm. Che et al. (2016) imputes the missing values using only the latest measurements, the mean value of each stream, and the time interval. It is not bi-directional and so cannot use information available at a given time to update estimates of information that is missing at an earlier time.

4 DEEP SENSING: ALGORITHM

In this section, we describe the training and runtime stages of the Deep Sensing algorithm. Fig. 2 shows block diagrams of the two stages.

4.1 TRAINING STAGE

The training stage of Fig. 2 shows the block diagram of the entire 5 blocks of the training stage. The first 4 blocks train Interpolation block (Φ), Imputation block (Ψ), Error Estimation block (Γ), Pre-
The objective of the interpolation and imputation blocks is to minimize the error that would be made in estimating missing measurements. Evidently, we cannot estimate the error of a measurement that is truly missing in the dataset. Instead we fix a measurement that was actually made, remove that measurement, form an estimate for the measurement, and then compute the error between the estimate and the actual measurement (that was deleted) using only the data set \( D - x^d \) (i.e. the data set with \( x^d \) removed). If \( x^d \) is an actual measurement and \( \hat{x}^d \) is the estimate formed when \( x^d \) is removed then the loss can be defined as the mean squared error (MSE) \( l(\hat{x}^d, x^d) = (\hat{x}^d - x^d)^2 \). The loss for the entire dataset \( D \) is defined as

\[
L(\{\hat{x}^d, x^d\}) = \sum_{n=1}^{N} \left[ \frac{\sum_{t=1}^{T_n} \sum_{d=1}^{D} m^d(n) \times (\hat{x}^d(n) - x^d(n))^2}{\sum_{t=1}^{T_n} \sum_{d=1}^{D} m^d(n)} \right].
\]

Interpolation: The objective of the interpolation block is to construct an interpolation function \( \Phi \) that operates within a stream. To emphasize that the estimate for \( x^d \) depends on the data with \( x^d \) removed, we abuse notation and write \( \hat{x}^d = \Phi(D - x^d) \). However, keep in mind that we are actually only using the data from stream \( d \), not the data from other streams. We construct the interpolation function \( \Phi \) using a bi-directional recurrent neural network (Bi-RNN) with a Gated Recurrent Unit (GRU). However, unlike a conventional Bi-RNN (Graves & Schmidhuber (2005)), the timing of inputs into the hidden layer is lagged in the forward direction and advanced in the backward direction: at time \( t \), inputs of forward hidden states come from \( t-1 \) and inputs of backward hidden states come from \( t+1 \). Mathematically, we have:

\[
\begin{align*}
o_t &= \overrightarrow{W}_h \overrightarrow{h}_t + \overrightarrow{U}_h \overrightarrow{h}_t + c, \\
\overrightarrow{h}_t &= (1 - \overrightarrow{z_t}) \odot \overrightarrow{h}_{t-1} + \overrightarrow{z_t} \odot \overrightarrow{h}_t, \\
\overrightarrow{z}_t &= \sigma(\overrightarrow{W}_z x_{t-1} + \overrightarrow{U}_z \overrightarrow{h}_{t-1} + \overrightarrow{V}_z \delta_{t-1} + \overrightarrow{c}_z), \\
\overrightarrow{h}_t &= \sigma(\overrightarrow{W}_h x_{t-1} + \overrightarrow{U}_h (\overrightarrow{r}_t \odot \overrightarrow{h}_{t-1}) + \overrightarrow{V}_h \delta_{t-1} + \overrightarrow{c}_h), \\
\overrightarrow{r}_t &= \sigma(\overrightarrow{W}_r x_{t-1} + \overrightarrow{U}_r \overrightarrow{h}_{t-1} + \overrightarrow{V}_r \delta_{t-1} + \overrightarrow{c}_r)\end{align*}
\]

where \( \odot \) is element-wise multiplication, \( \sigma \) is the sigmoid function, \( \phi \) is tanh function, and arrows indicate forward/backward direction. The output \( o_t \) is the interpolated value \( \hat{x}_t \). In this interpolation
block, we are only using/capturing the temporal correlation within each stream. As a consequence, the matrices $U, V, W$ are block-diagonal. Hence the total number of parameters that must be learned is linear in the number of streams $D$ (on the other hand, in standard Bi-RNN, the number of parameters to be learned will be on the order of $D^2$). This avoids overfitting and leads to significant performance improvements as compared to standard Bi-RNN. See the Interpolation part of Fig. 3.

**Imputation:** The objective of the imputation block is to construct an imputation function $\Psi$ that operates across streams. Again, we abuse notation and write $\hat{x}_{dt} = \Psi(D - x_{dt})$. Keep in mind that now we are using only data at time stamp $t$, not data from other time stamps. We construct the function $\Psi$ to be independent of the time stamp $t$; so we construct it using fully connected layers (FC); see Imputation part of Fig 3:

$$o_t = Wh_t + c_o,$$
$$h_t = Ux_t + V\hat{x}_t + Qm_t + c_h$$

where $o_t = \hat{x}_t$ and the block-diagonal entries of $U$ are zero because we do not use $x_{dt}$ to estimate $\hat{x}_{dt}$. We use multiple deeply stacked FC layers using linear activation functions.

We jointly learn the functions $\Phi$ and $\Psi$ using the stacked networks of Bi-RNN and FC layers.

$$\Phi^*, \Psi^* = \arg \min_{\Phi, \Psi} \mathcal{L}(\{\Phi\{x_{dt}, \Phi\{x_{dt}, m_t, \delta_{dt}\}_{t=1:T}, m_d\}_{d=1:D}, x_{dt}\})$$

We refer to the entire structure as a Multi-directional Recurrent Neural Network (M-RNN); see Fig 3.

**Prediction:** Now that we have a procedure to reconstruct (interpolate/impute) missing data, we use the reconstructed data to predict the labels. We accomplish this in the prediction block. Once again, we construct the function to minimize the prediction error when we predict an observed label. The loss function is defined as $\mathcal{L}(\hat{y}_t, y_t) = \frac{1}{N} \sum_{n=1}^{N} \sum_{t=1}^{T} (\hat{y}_{i(n)}(n) - y_{i(n)})^2$ The prediction block optimizes the function:

$$\Omega^* = \arg \min_{\Omega} \mathcal{L}(\hat{y}_t, y_t) = \arg \min_{\Omega} \mathcal{L}(\{\hat{x}_{dt}, \hat{x}_{dt}, \delta_{dt}\}_{t=1:T, d=1:D}, y_t)$$

Note that we use the time intervals $\delta_{dt}$ as inputs to the prediction function in order to deal with the fact that the data streams are irregularly sampled. This optimization problem is a standard problem for timely prediction so we can use a standard GRU-based RNN (Chung et al. (2014)) to solve it.
Error Estimation: At runtime, we have to decide when/what to sample in the active sensing block. We make this decision on the basis of predictive gain which is determined by the difference between our estimate of a measurement and what the actual value of the measurement would be; the actual error \( e_t^d = |\hat{x}_t^d - x_t^d| \). Of course, we do not know what this will be because we do not know what the actual value of the measurement would be. Hence we need an estimate \( \hat{e}_t^d \) of the actual error. As before, we construct this estimate on the basis of the actual training data that we have. For tractability, we posit that this estimate depends on the pattern of missing data and on time intervals of the measurements but not on actual measurements. We use the same mean square loss function:

\[
\mathcal{L}(\{\hat{e}_t^d, e_t^d\}) = \frac{N}{\sum_{t=1}^{T_n} \sum_{d=1}^{D} m_t^d(n) \times (\hat{e}_t^d(n) - e_t^d(n))^2}{\sum_{t=1}^{T_n} \sum_{d=1}^{D} m_t^d(n)}.
\]

Hence we need to solve for the function

\[
\mathbf{\Gamma}^* = \arg \min_{\mathbf{\Gamma}} \mathcal{L}(\{\hat{e}_t^d, e_t^d\}) = \arg \min_{\mathbf{\Gamma}} \mathcal{L}(\{\mathbf{m}_t^d\}_{d=1:D}, \{\delta_t^d, \tau_t^d\}_{t=1:T}, e_t^d).
\]

Because this involves both inter-stream and intra-stream variables, we again use our M-RNN structure. However, the inputs and outputs are different: for the interpolation and imputation blocks, the inputs are the measurements \( \{x_t^d\} \), sensing information \( \{m_t^d\} \) and the time intervals \( \{\delta_t^d\} \) and the output is the estimated measurement \( \hat{x}_t^d \). For the error estimation block, the inputs are the sensing information \( \{m_t^d\} \) and the time intervals \( \{\delta_t^d\} \) and the output is the estimation error \( \hat{e}_t^d \).

Adaptive Sampling: As pointed out in Section 2, the decision of what/when to sense arises from trading off the cost of measurement against the predictive gain of measurement. To this point, we have constructed a procedure that achieves a certain performance – predictive gain – at a prescribed cost. If we are willing to settle for a lower level of performance, we can do so at lower cost by sampling less. To know how much less to sample we need to know how much information would be lost if we sampled less, which we can determine by carrying out the previous procedure to produce multiple representations based on different patterns of missing data. For each representation we need to train the functions \( \Phi, \Psi, \Omega, \Gamma \) on the appropriate training set, which is smaller than the original training set.

To construct these representations, we begin with the original training set and remove additional measurements. We should not do this at random, but rather using the informational criteria we use to decide on active sensing at runtime: remove measurements whose predictive gain is below a given threshold. We call this adaptive sampling. This will yield a decreasing sequence of data sets \( D_0, D_1, ..., D_R \) (where \( D_0 = D \), the original dataset).

Fix thresholds \( u_1, ..., u_R > 0 \). (In practice, these would be specified by the user.) We begin with \( D_0 \). For each measurement \( x_t^d(n) \in D_0 \) we use the current functions \( \Phi = \Phi_0, \Psi = \Psi_0, \Omega = \Omega_0, \Gamma = \Gamma_0 \) to compute the predictive gain from that measurement in the current dataset. We sequentially delete all the measurements whose predictive gain is below the prescribed threshold \( u_1 \); this yields a reduced data set \( D_1 \). We now retrain on \( D_1 \) to obtain new functions \( \Phi_1, \Psi_1, \Omega_1, \Gamma_1 \) and repeat the same procedure: for each measurement \( x_t^d(n) \in D_1 \), we compute the predictive gain from that measurement in the current dataset and sequentially delete those measurements whose predictive gain is below threshold \( u_2 \), etc. In retraining to obtain the functions \( \Phi_r, \Psi_r, \Omega_r, \Gamma_r \), we initialize the parameters of each block to be the optimal parameters from the previous dataset \( D_{r-1} \); this leads to fast convergence. (For the original dataset, we use the Xavier initialization method.)

Note that if we increase the threshold \( u_r \), we delete more data and retain fewer samples, so the expenditure on sampling is smaller. However because we have trained on fewer samples, our predictions will be less accurate. This creates the cost-performance trade-off.

4.2 Runtime

The runtime stage of Fig. 2 shows the block diagram in real-time. At runtime, the user prescribes an acceptable level of performance (required predictive gain and a confidence level \( \lambda \)). We then choose the largest \( r \) for which the representation \( \mathbf{\Lambda}_r = (\Phi_r, \Psi_r, \Omega_r, \Gamma_r) \) achieves at least the prescribed level of performance and use \( \mathbf{\Lambda}_r \) to make the sampling decision. As described in Section 2, this means that at time \( s_T \) we choose \( s_{T+1} \) to be the first time at which the predictive gain (estimated by using \( \mathbf{\Gamma}_r, \Omega_r \) and the prescribed confidence level) is above the given cost.
At current time $t$, Deep Sensing estimates the measurements at time $t + 1$ using only measurements until the current time $t$, setting all other measurements to their initialization value. Then, it computes the predictive gain of each stream and senses the streams whose predictive gain is larger than the threshold $u_r$. After sampling measurements at time $t + 1$, these newly sampled measurements are used to re-estimate the measurements that were missing before $t + 1$ (using the backward M-RNN architecture). Then, using all the newly imputed measurements through time $t + 1$, Deep Sensing makes its prediction at $t + 1$. (If at time $t + 1$ there are no streams for which the predictive gain is larger than the threshold, Deep Sensing moves to time $t + 2$, etc. Notice that Deep Sensing is only required to issue a prediction at each time stamp; i.e. at each time at which a measurement is actually made.) Figure 5 in the Appendix details the operation of Deep Sensing in runtime. Deep Sensing is also computationally efficient. For instance, on the MIMIC-III dataset (23,200 samples, 40 dimensions, 25 time stamps), Deep Sensing takes less than one hour on a machine with i7-6900K CPU (3.2GHz x 16) and 64GB RAM.

5 EXPERIMENTS

In this section, we evaluate the performance of Deep Sensing using two real-world medical datasets. Our experimental results present three sets of comparisons: active sensing, prediction, and missing value inference. The first comparison shows the performance gain of Deep Sensing (in comparison with benchmarks) in sensing the critical measurements for prediction. The second and third comparisons show the performance gain of M-RNN algorithm in estimating missing values and the effect on prediction accuracy. We describe all configurations of the various algorithms in the Appendix.

5.1 DATA DESCRIPTION

We conducted our experiments on the two real-world medical datasets. First, we used MIMIC-III (Johnson et al. (2016)) which monitored the patients in the intensive care units (ICU) to timely estimate the mortality risk. We used 22,803 patients who were monitored for 20 vital signs (e.g. heart rate, respiratory rate) and 20 lab tests (e.g. creatinine, chloride). So there are 40 physiological data streams in all. The sampling times of vital signs and lab tests are around 1 hour and 24 hours, respectively. Second, we used a dataset (that we call Wards) that was assembled and described by Alaa et al. (2017) and which the authors were kind enough to share with us. This dataset monitored patients in hospital wards in order to make timely predictions of clinical deterioration. This dataset has data for 6,094 patients with 37 physiological data streams including both vital signs and lab tests. The sampling times of vital signs and lab tests are approximately 4 hours and 24 hours, respectively.

5.2 SIMULATION SETUP

We randomly divided the dataset into a mutually exclusive training set (80%) and testing set (20%). We conducted 10 independent experiments with different combinations of training/testing sets; we report the mean and variance of the performance in the 10 experiments. (For details, see the Appendix.) In our experiments we are trying to predict which patients will experience an adverse event within 24 hours from the current time, so we assigned the label 1 to patients who experienced the relevant adverse event (death or clinical deterioration) within 24 hours; for other patients we assigned the label 0. (Formally: $y_t = 1$ for $s_t > S_T - 24$ and $y_t = 0$ for $s_t \leq S_T - 24$ where $S_T$ is the time that the adverse event occurred.)

Active sensing: To evaluate the performance of Deep Sensing, we graph cost (proxied here as the observation rate) versus predictive accuracy (area under the ROC curve (AUC)) for Deep Sensing (with multiple representations), Deep Sensing (with a single representation – using only the original dataset to train), Deep Sensing (with random sampling) and two benchmarks based on the method of Che et al. (2016) for prediction with missing data (sampling either using the method of Deep Sensing or randomly). We take the cost for lab tests to be 5 times the cost for taking vital signs.

Prediction: We also evaluate prediction given only the available observations. The prediction we consider is the adverse events (death or clinical deterioration); we use AUC as the performance metric. We compare the performance of Deep Sensing with three state-of-the-art RNN timely-prediction models and a GRU-based RNN method using conventional estimation methods for interpolation and imputation. To make the comparison fair, we use GRU-based RNNs for each benchmark. Deep
Figure 4: Active Sensing: AUC vs Cost for Different Solutions with MIMIC-III dataset (Lab test cost = 5 × Vital sign cost)

Table 1: AUC for Deep Sensing and Benchmarks with MIMIC-III dataset (See the text for descriptions of Settings A, B). *: p-value < 0.05

<table>
<thead>
<tr>
<th>Proposed Model</th>
<th>Deep Sensing</th>
<th>MIMIC-III (Setting A)</th>
<th>MIMIC-III (Setting B)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.8019 ± 0.0112 (-)</td>
<td>0.8019 ± 0.0112 (-)</td>
<td></td>
</tr>
<tr>
<td>RNN based</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Choi et al. [2015]*</td>
<td>0.7112 ± 0.0134 (31.4 %)</td>
<td>0.7598 ± 0.0110 (17.5 %)</td>
<td></td>
</tr>
<tr>
<td>Lipton et al. [2016]*</td>
<td>0.7072 ± 0.0108 (32.3 %)</td>
<td>0.7551 ± 0.0115 (19.1 %)</td>
<td></td>
</tr>
<tr>
<td>Che et al. [2016]*</td>
<td>0.7133 ± 0.0111 (30.9%)</td>
<td>0.7593 ± 0.0123 (17.7 %)</td>
<td></td>
</tr>
<tr>
<td>Interpolation + RNN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Spline*</td>
<td>0.7045 ± 0.0137 (33.0 %)</td>
<td>0.7542 ± 0.0108 (19.4 %)</td>
<td></td>
</tr>
<tr>
<td>Cubic*</td>
<td>0.7012 ± 0.0129 (33.7 %)</td>
<td>0.7569 ± 0.0112 (18.5 %)</td>
<td></td>
</tr>
<tr>
<td>Imputation + RNN</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MICE*</td>
<td>0.7093 ± 0.0132 (31.9 %)</td>
<td>0.7571 ± 0.0121 (18.4 %)</td>
<td></td>
</tr>
<tr>
<td>Kernel*</td>
<td>0.7002 ± 0.0119 (33.9 %)</td>
<td>0.7534 ± 0.0139 (19.7 %)</td>
<td></td>
</tr>
<tr>
<td>EM*</td>
<td>0.7019 ± 0.0098 (33.5 %)</td>
<td>0.7531 ± 0.0107 (19.8 %)</td>
<td></td>
</tr>
</tbody>
</table>

Sensing is compared with the benchmarks in two settings. In setting A, we sampled 60% of the measurements; for Deep Sensing, we used the Deep Sensing algorithm, for the benchmarks, we use random sampling. In setting B we sampled 60% of the measurements, using the Deep Sensing algorithm everywhere.

Estimation of missing values: To evaluate the performance of the M-RNN algorithm (the combination of interpolation and imputation block) in estimating missing values, we compare with other standard methods: interpolations (Spline and Cubic [Kreindler & Lumsden 2012]), imputations (MICE [White et al. 2011]), Kernel (Rehfeld et al. 2011) and EM (García-Laencina et al. 2010)). We randomly remove 30% of the observations and treat them as missing. We then estimate the missing observations values using our M-RNN algorithm and benchmarks. We use the root mean square error (RMSE) between estimated values and actual observed values as the performance metric.

5.3 Simulation Results and Discussion

Active Sensing: As Fig. 4 illustrates, Deep Sensing predicts best for every specification of cost; equivalently, Deep Sensing expends the least cost for every specified prediction accuracy. Fig. 4 also shows that the performance gains achieved by Deep Sensing come both from active sampling and from better inference. As seen in Figure 4, if the observation rate were 100% (so there would be no gain from active sensing), the AUC improvement would be limited. However, as the observation rate decreases the AUC gain increases because Deep Sensing actively decides what to sample and when to sample, thereby providing results that are much superior to random sampling.

Prediction: Table 1 and 2 provide the mean, standard deviation, and performance gain (%) (in terms of AUC) from Deep Sensing in comparison to the benchmarks for two real-world medical datasets. Table 1 and 2 show that Deep Sensing provides significant gains of the prediction accuracy for both datasets (around 30% in Setting A and 20% in Setting B for all the benchmarks).
Table 2: AUC for Deep Sensing and Benchmarks with Wards dataset (See the text for descriptions of Settings A, B). *: p-value < 0.05

<table>
<thead>
<tr>
<th>Proposed Model</th>
<th>Deep Sensing</th>
<th>Wards (Setting A)</th>
<th>Wards (Setting B)</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>RNN based</em></td>
<td>Choi et al. (2015)*</td>
<td>0.7739 ± 0.0264 (26.9 %)</td>
<td>0.8028 ± 0.0184 (16.2 %)</td>
</tr>
<tr>
<td>Lipton et al. (2016)*</td>
<td>0.7893 ± 0.0237 (21.6 %)</td>
<td>0.8107 ± 0.0191 (12.7 %)</td>
<td></td>
</tr>
<tr>
<td>Che et al. (2016)*</td>
<td>0.7905 ± 0.0143 (21.1 %)</td>
<td>0.8159 ± 0.0160 (10.3 %)</td>
<td></td>
</tr>
<tr>
<td>Interpolation</td>
<td>Spline*</td>
<td>0.7829 ± 0.0085 (23.9 %)</td>
<td>0.8023 ± 0.0187 (16.4 %)</td>
</tr>
<tr>
<td>+ RNN</td>
<td>Cubic*</td>
<td>0.7712 ± 0.0084 (27.8 %)</td>
<td>0.7993 ± 0.0137 (17.7 %)</td>
</tr>
<tr>
<td>Imputation</td>
<td>MICE*</td>
<td>0.7499 ± 0.0096 (33.9%)</td>
<td>0.7877 ± 0.0149 (22.2 %)</td>
</tr>
<tr>
<td>+ RNN</td>
<td>Kernel*</td>
<td>0.7397 ± 0.0155 (36.5 %)</td>
<td>0.7728 ± 0.0187 (27.3 %)</td>
</tr>
<tr>
<td></td>
<td>EM*</td>
<td>0.7593 ± 0.0168 (31.4%)</td>
<td>0.7784 ± 0.0163 (25.5%)</td>
</tr>
</tbody>
</table>

Significant gains for prediction come from the combination of accurate missing value inference and active sensing as seen in Figure 4.

**Estimation of missing values:** Table 3 shows the mean and standard deviation of the RMSE for M-RNN and benchmarks for both the MIMIC-III and the Wards dataset. The RMSE of M-RNN is less than half that of the best benchmark in MIMIC-III dataset and less than 70% that of the best benchmark in Wards dataset. All the improvements are statistically significant (p-value < 0.05).

Table 3: RMSE of Missing information for M-RNN and Benchmarks with MIMIC-III and Wards datasets. *: p-value < 0.05

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Metrics</th>
<th>Interpolation</th>
<th>Imputation</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>M-RNN Spline</td>
<td>Cubic MICE Kernel EM</td>
</tr>
<tr>
<td>MIMIC-III</td>
<td>RMSE - Mean</td>
<td><strong>0.0137</strong></td>
<td>0.0735* 0.0279* 0.0611* 0.0556* 0.0467*</td>
</tr>
<tr>
<td></td>
<td>RMSE - Std</td>
<td><strong>0.0013</strong></td>
<td>0.0012 0.0013 0.0011 0.0011 0.0014</td>
</tr>
<tr>
<td>Wards</td>
<td>RMSE - Mean</td>
<td><strong>0.0169</strong></td>
<td>0.0314* 0.0211* 0.0554* 0.0627* 0.0761*</td>
</tr>
<tr>
<td></td>
<td>RMSE - Std</td>
<td><strong>0.0019</strong></td>
<td>0.0011 0.0021 0.0013 0.0014 0.0017</td>
</tr>
</tbody>
</table>

6 Conclusion

The problem of active sensing is a very important one but has not been thoroughly treated in the literature. We present here a solution based on a novel deep learning architecture. As part of the solution, we provide a new method for reconstructing missing data that exploits joint interpolation within data streams and imputation across data streams. We demonstrate that Deep Sensing makes large and statistically significant improvements in comparison with state-of-the-art benchmarks in two real-world datasets.
REFERENCES


APPENDIX

THE DETAIL PROCEDURES IN RUNTIME OF DEEP SENSING

Figure 5: The detail procedures in runtime of Deep Sensing
CONFIGURATIONS OF THE EXPERIMENTS

We used ‘interp1’ package in MATLAB for interpolation algorithms, and ‘mice’ and ‘amelia’ packages in R for imputation algorithms. Table 4 illustrates the details of the prediction algorithm.

<table>
<thead>
<tr>
<th>Blocks</th>
<th>Categories</th>
<th>Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Interpolation</td>
<td>Model: Modified Bi-RNN with GRU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initialization: Xavier Initialization (Glorot &amp; Bengio (2010))</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optimization: Adam* Optimization (Kingma &amp; Ba (2014)) (learning rate = 0.05)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Batch size and iterations: Batch size = 100, Iterations = 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Depth: 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Constraints: The matrix parameters are block-diagonals</td>
</tr>
<tr>
<td></td>
<td>Imputation</td>
<td>Model: Fully Connected Layers</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initialization: Xavier Initialization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optimization: Adam Optimization (learning rate = 0.05)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Batch size and iterations: Batch size = 100, Iterations = 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Depth: 10 (Activate function: Linear)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Constraints: The block-diagonal part of the matrix is zero.</td>
</tr>
<tr>
<td></td>
<td>Error Estimation</td>
<td>Model: Multi-directional RNN (M-RNN) with GRU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initialization: Xavier Initialization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optimization: Adam Optimization (learning rate = 0.05)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Batch size and iterations: Batch size = 100, Iterations = 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Depth: 5 for Bi-RNN, 10 for FC with linear activation function</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Constraints: Block-diagonal matrix parameters for Bi-RNN</td>
</tr>
<tr>
<td></td>
<td>Prediction</td>
<td>Model: RNN with GRU</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initialization: Xavier Initialization</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Optimization: Adam Optimization (learning rate = 0.05)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Batch size and iterations: Batch size = 100, Iterations = 1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Depth: 10</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Constraints: None</td>
</tr>
</tbody>
</table>

Adam*: Adaptive Moment Estimation