Pay attention and you won't lose it: A deep learning approach to sequence imputation

Ilia Sucholutsky [Corresp., 1] , Apurva Narayan [2, 3] , Matthias Schonlau [1] , Sebastian Fischmeister [2]

[1] Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, Ontario, Canada

[2] Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada

[3] School of Engineering, University of British Columbia, Kelowna, British Columbia, Canada

Corresponding Author: Ilia Sucholutsky
Email address: isucholu@uwaterloo.ca

In most areas of machine learning, it is assumed that data quality is fairly consistent between training and inference. Unfortunately, in real systems, data are plagued by noise, loss, and various other quality reducing factors. While a number of deep learning algorithms solve end-stage problems of prediction and classification, very few aim to solve the intermediate problems of data pre-processing, cleaning, and restoration. Long Short-Term Memory (LSTM) networks have previously been proposed as a solution for data restoration, but they suffer from a major bottleneck: a large number of sequential operations. We propose using attention mechanisms to entirely replace the recurrent components of these data-restoration networks. We demonstrate that such an approach leads to reduced model sizes by as many as 2 orders of magnitude, a 2-fold to 4-fold reduction in training times, and 95% accuracy for automotive data restoration. We also show in a case study that this approach improves the performance of downstream algorithms reliant on clean data.

Pay Attention And You Won't Lose It: A deep learning approach to sequence imputation

Ilia Sucholutsky[1]**, Apurva Narayan**[2,3]**, Matthias Schonlau**[1]**, and Sebastian Fischmeister**[2]

[1]**Department of Statistics and Actuarial Science, University of Waterloo, Waterloo, Ontario, Canada**
[2]**Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, Ontario, Canada**
[3]**School of Engineering, University of British Columbia, Kelowna, British Columbia, Canada**

Corresponding author:
Ilia Sucholutsky[1]

Email address: isucholu@uwaterloo.ca

ABSTRACT

In most areas of machine learning, it is assumed that data quality is fairly consistent between training and inference. Unfortunately, in real systems, data are plagued by noise, loss, and various other quality reducing factors. While a number of deep learning algorithms solve end-stage problems of prediction and classification, very few aim to solve the intermediate problems of data pre-processing, cleaning, and restoration. Long Short-Term Memory (LSTM) networks have previously been proposed as a solution for data restoration, but they suffer from a major bottleneck: a large number of sequential operations. We propose using attention mechanisms to entirely replace the recurrent components of these data-restoration networks. We demonstrate that such an approach leads to reduced model sizes by as many as 2 orders of magnitude, a 2-fold to 4-fold reduction in training times, and 95% accuracy for automotive data restoration. We also show in a case study that this approach improves the performance of downstream algorithms reliant on clean data.

INTRODUCTION

The growth of the Internet of Things has led to a sharp increase in the number of real-world datasets being collected, processed, and analyzed in a decentralized way. Unfortunately such data are rarely as clean as in lab conditions and are often plagued by noise, loss, and various other quality reducing factors. However, in many areas of machine learning, algorithms are still designed with the assumption that data at time of inference is of the same quality as training data. As a result, when algorithms are trained on clean, pre-processed samples, they may fail to function as desired when performing inference on raw data. In particular, data loss is a serious issue for many algorithms. This is especially important for safety-critical systems like cars, where a failure can have very serious consequences for users. Anomaly detection, and other end-stage algorithms, need to have very high performance in such systems, but lost data may be a large factor in preventing this. As a result, for systems that analyze streams of data, a method is needed to quickly impute the missing values as closely as possible to what the true values would have been.

Deep learning techniques have been proven to be effective at learning temporal structure of data [LeCun et al., 2015]. However, while a number of deep learning algorithms have been successfully shown to solve numerous end-stage problems like prediction and classification [Glorot et al., 2011, LeCun et al., 2015, Abadi et al., 2016], very few attempts have been made to use them for solving the intermediate problems of data pre-processing [Kotsiantis et al., 2006, García et al., 2015], cleaning [Kotsiantis et al., 2006, García et al., 2015], and restoration [Efron, 1994, Lakshminarayan et al., 1996], even though from a machine learning perspective these end-stage and intermediate problems can be very similar. Long

Short-Term Memory (LSTM) networks have previously been proposed as a solution to these intermediate problems [Zhou and Huang, 2017, Sucholutsky et al., 2019], but they suffer from major bottlenecks like requiring large numbers of sequential operations that cannot be parallelized. Recently, Transformer [Vaswani et al., 2017], a novel encoder-decoder model that heavily uses attention mechanisms [Luong et al., 2015], was proposed as a replacement for encoder-decoder models that use LSTM or convolutional layers, and was shown to achieve state-of-the-art translation results with orders of magnitude fewer parameters than existing models. Inspired by this impressive result, we propose using a version of Transformer with modified hyper-parameters, that we will refer to in this paper as "Restorer". Restorer solves the intermediate problem of restoring missing elements in sequences of discrete data while entirely replacing the recurrent components of existing solutions. We demonstrate that such an approach leads to reduced model sizes, faster training times, and higher-quality reconstruction when compared to what we will refer to as "the LSTM model" described by Sucholutsky et al. [2019].

Our contributions are as follows:

- We implement and empirically validate the neighbourhood-restriction on Transformer that was theoretically proposed by Vaswani et al. [2017].

- We propose and create Restorer, a Transformer-based model tailored for imputing sequential data.

- We reproduce the LSTM model and restoration results from Sucholutsky et al. [2019] in Keras

- We match the state-of-the-art (SOTA) accuracy in restoring lost automotive CAN data with a model that has just 0.6% of the number of parameters of the previous SOTA model

- We improve the SOTA accuracy in restoring lost automotive CAN data by up to 7% while requiring 2-4 times less training time than the previous SOTA model.

- We demonstrate Restorer's positive impact on downstream algorithm performance in a case study where it outperforms both the benchmark and LSTM models.

The rest of this paper is divided into four sections. Section 1 will discuss existing work. Section 2 will describe the Restorer and its components. Section 3 will detail the experimental results. Finally, Section 4 will discuss some conclusions and future steps.

1 BACKGROUND

1.1 Sequence Modelling with Deep Learning

The most popular deep learning architecture for sequence modelling is Recurrent Neural Networks (RNNs), a type of neural network with an internal feedback mechanism that can be used as a form of memory [Williams and Zipser, 1989]. Probably the most successful extension of RNNs are LSTMs, which increase the flexibility of the internal feedback mechanism [Gers and Schmidhuber, 2001]. More recently, LSTM-based encoder-decoder models like Seq2Seq were shown to improve the state-of-the-art in sequence modelling and the addition of attention mechanisms was shown to further increase their performance [Sutskever et al., 2014]. In such models, the encoder performs an embedding of the entire input sequence before the decoder begins to use this embedding as input when generating an output sequence.

There has recently been much discussion about whether recurrent models provide any intrinsic advantage over feed-forward models [Miller and Hardt, 2018]. For example, Vaswani et al. [2017] recently proposed that an encoder-decoder model they call Transformer, can be built that entirely eschews recurrent components. Transformer makes use of several attention mechanisms to form an architecture that significantly outperforms the state-of-the-art on machine translation tasks without resorting to recurrent layers [Vaswani et al., 2017]. While Transformer was demonstrated to work specifically with the task of machine translation, its impressive performance suggests that it is not unreasonable to expect it can be adapted to work with other sequence modelling tasks.

1.2 Data Restoration with Deep Learning

Classical techniques for imputing lost values typically revolve around using a local or global mean wherever a missing value occurs [Donders et al., 2006, Schmitt et al., 2015]. When working with discrete

94 data, instead of means, class values that occur with high frequency locally or globally are often used in
95 techniques like hot-deck imputation [Andridge and Little, 2010, Myers, 2011, Aljuaid and Sasi, 2016],
96 k-nearest neighbours [Cover and Hart, 1967], decision trees [Safavian and Landgrebe, 1991], etc. Other,
97 more complex, techniques work only with multivariate, and primarily continuous, data [Raghunathan
98 et al., 2001, Buuren and Groothuis-Oudshoorn, 2010]. However, all of these techniques fail to properly
99 address the problem of imputing missing discrete data in sequences [Schafer and Graham, 2002].

100 Interestingly, while deep learning has been repeatedly shown to be effective at sequential modelling,
101 there has been little work in applying it to data restoration. Of the methods that do address data restoration,
102 many are designed specifically for the restoration of continuous data [Duan et al., 2014, Zhou and Huang,
103 2017, Niklaus et al., 2017]. Some methods are intended for use only with image data through techniques
104 like in-painting [Xie et al., 2012, Lehtinen et al., 2018, Altinel et al., 2018]. More recently, Generative
105 Adversarial Networks (GANs) have been used for super-resolution and denoising of images, although
106 their primary use case is typically image generation [Goodfellow et al., 2014, Ledig et al., 2017]. There
107 are also a number of data restoration methods that assume that all the data is available at once [Blend and
108 Marwala, 2008, Leke et al., 2015, Gondara and Wang, 2017, Beaulieu-Jones and Moore, 2017]. However,
109 in the context of data restoration specifically for discrete, streaming data it was only recently demonstrated
110 that a simple LSTM model can be used to restore missing message IDs in automotive data [Sucholutsky
     et al., 2019].

Table 1. Comparison of different layers where n is sequence length, d is dimension of representation, k is kernel size, and r is neighbourhood size [Vaswani et al., 2017].

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

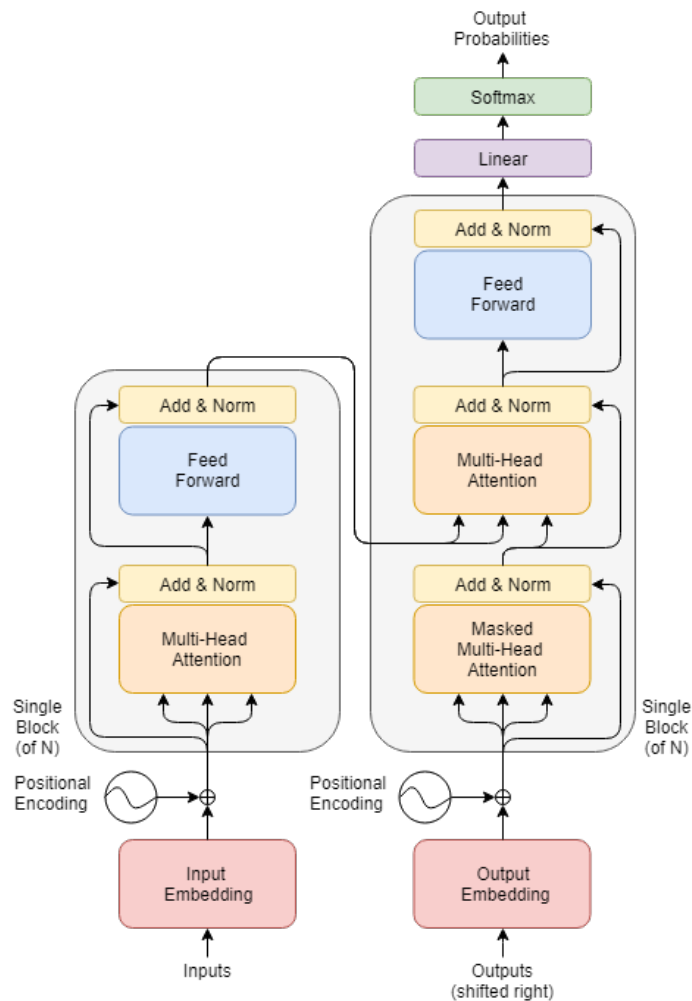
111

2 RESTORER

2.1 Attention

114 In an RNN, sequences are represented within their hidden states so the output of each hidden state needs to
115 be computed before the model can access the next step in the sequence. This results in a large bottleneck
116 since a number of sequential operations have to be performed that cannot be parallelized. In addition, this
117 results in long path lengths that elements of the sequence have to traverse, which has been shown to be
118 undesirable behaviour when trying to model long-term dependencies [Hochreiter et al., 2001]. Meanwhile,
119 not only can convolutional layers be computationally expensive [Chollet, 2017], but they also need to be
120 stacked in order to connect all inputs to all outputs [Kalchbrenner et al., 2016], which in turn increases
121 path length.

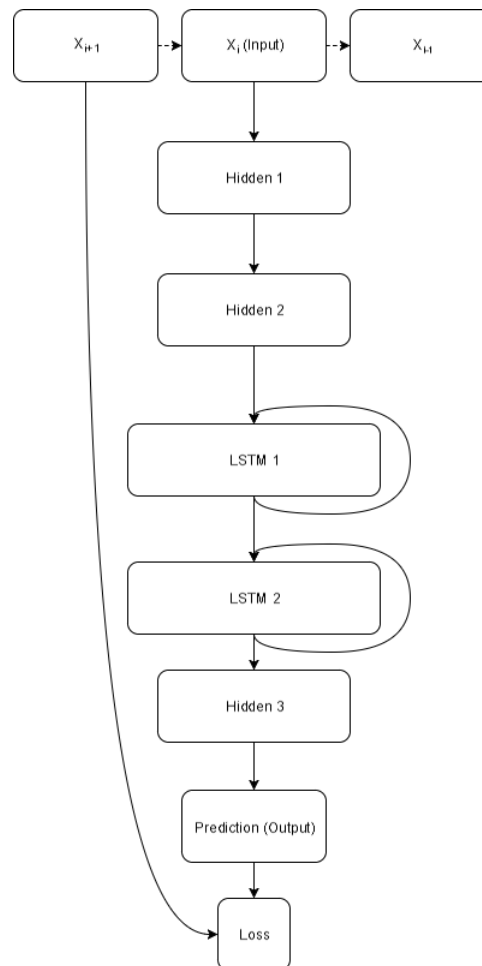
122 Both recurrent and convolutional layers can instead be replaced with self-attention layers as described
123 in the Transformer model in Vaswani et al. [2017]. This removes the bottleneck of sequential operations
124 and reduces path length as dot-product attention provides access to the entire history at once. As a
125 result, training time should be decreased, while the ability to learn long-term dependencies increased. We
126 confirm that this result holds in practice by demonstrating it empirically in Section 3. This improvement
127 does come at a cost: an increase in complexity relative to sequence length since the entire sequence is
128 operated on at once and every element attends to every other input. To avoid this new bottleneck, a version
129 of Transformer can be used where input is restricted to neighbourhoods of size r instead of operating on
130 the entire sequence at once [Vaswani et al., 2017]. Table 1 summarizes the theoretical differences between
131 these layer types across three key metrics.

Figure 1. Transformer Model Architecture [Vaswani et al., 2017]: The Transformer consists of an encoder and decoder each made up of N blocks. Input is a sequence of events, output is a sequence of predicted events.

2.2 Architecture

Restorer is not an RNN or LSTM; it instead follows the Transformer architecture illustrated in Figure 1 but with modified hyper-parameters as seen in Table 2. For comparison, the LSTM architecture is pictured in Figure 2. For the various Restorer model versions, N_{blocks} is the number of blocks in the encoder and decoder, d_{model} is the dimension of the embedding and consequently is used as the scaling factor in our scaled dot product attention (instead of d_k as described in Vaswani et al. [2017]), d_k is the dimension that keys in attention are projected to, d_v is the dimension that values in attention are projected to, d_h is the size of the hidden layers, and n_{head} is the number of parallel heads in each multi-head attention layer. For the LSTM model, n_{hid} is the number of non-LSTM hidden layers, n_{lstm} is the number of LSTM layers, d_h is the size of the non-LSTM hidden layers, d_{lstm} is the size of LSTM layers, and n_{steps} is the number of steps for which the LSTM is unrolled.

As mentioned above, one of the theoretical advantages of the proposed approach, is a large reduction in the number of trainable parameters. In fact, we show empirically that our Restorer architectures have as many as two orders of magnitude fewer parameters than the LSTM model as seen in Table 2.

Sucholutsky et al. [2019] implemented the LSTM restoration model in Tensorflow. We have re-implemented it in Keras in order to be more consistent as we implement our Restorer model in Keras based on the implementation of Transformer by Lsdefine [2018].

Figure 2. LSTM Model Architecture [Sucholutsky et al., 2019]: The LSTM model consists of two hidden layers followed by two recurrent LSTM layers and one additional hidden layer. Input is a sequence of events, output is a prediction of next event in the sequence. Loss is calculated as a logloss function comparing the true next event to the predicted one.

3 EXPERIMENTS

3.1 Data

For consistency during comparison, we utilized a similar dataset to the one that the LSTM restoration model was trained on [Sucholutsky et al., 2019]. We collected automotive Controller Area Network (CAN) traces from a Lexus RX450h hybrid SUV. A CAN trace is a long sequence of timestamped messages that consist of an ID and a payload of several bytes of data. We strip away the payload and timestamps and keep only the message IDs in order to obtain a dataset consisting of discrete, sequential data. In particular, 20 traces were collected of the same maneuver, the vehicle decelerating from 20km/h to 0km/h. The train/test split was kept at 75%/25% to be consistent with the experiments in Sucholutsky et al. [2019]. Each trace contains a sequence of about 6500 messages made up of 43 unique message IDs. In other words, we are presented with a sequence modelling task with a dictionary size of 43. Transformer-based models have already been shown to outperform LSTMs on classical sequence modelling tasks like language [Radford et al., 2018, 2019], music [Huang et al., 2018], and protein sequences [Rives et al., 2019]. The CAN dataset is instead representative of alternative tasks with datasets that have relatively few training samples but require very high accuracy for results to be useful.

Table 2. Size comparison of different model versions in terms of number of parameters

Model							params
LSTM	n_{hid}	n_{lstm}	d_h	d_{lstm}	n_{steps}		
	3	2	88	352	40		1,640,892
Restorer	N_{blocks}	d_{model}	d_k	d_v	d_h	n_{head}	
(main)	1	15	64	64	512	4	80,824
	2	15	64	64	512	4	158,873
	3	15	64	64	512	4	236,922
	5	15	64	64	512	4	393,020
	10	15	64	64	512	4	783,265
	15	15	64	64	512	4	1,173,510
	20	15	64	64	512	4	1,563,755
(1-head)	1	15	64	64	512	1	46,264
	2	15	64	64	512	1	89,753
	3	15	64	64	512	1	133,242
(mini)	1	15	32	32	88	1	14,216
	2	15	32	32	88	1	25,657
	3	15	32	32	88	1	37,098
(mini2)	1	32	16	16	32	1	16,704
	2	32	16	16	32	1	27,488
	3	32	16	16	32	1	38,272
(micro)	1	16	8	8	16	1	5,792
	2	16	8	8	16	1	8,624
	3	16	8	8	16	1	11,456

3.2 Benchmark

To better interpret the results of our experiments, we compare them against the same benchmark as Sucholutsky et al. [2019]. This benchmark can be described as either a fast Markov model or a history search, and in essense is a conditional probability maximization method. When there are constraints on data or computational power, Hidden Markov Models can match the performance of LSTMs [Panzner and Cimiano, 2016] so we believe this is a good benchmark that both the LSTM from Sucholutsky et al. [2019] and Restorer can be compared against.

Sucholutsky et al. [2019] describes the benchmark in detail, but in short, we select all possible n-grams [Schonlau et al., 2017, Lin et al., 2012, Lesher et al., 1999] from the training data and count the frequency with which they are followed by each event type. When we see an n-gram in the testing data followed by a missing event, we fill in the blank with the most frequently occuring event type corresponding to that n-gram. The iterative procedure from Sucholutsky et al. [2019] for collecting the frequencies is recreated in Algorithm 1.

Accuracy for this benchmark reaches its peak next-event prediction accuracy of 76.55% (30.07% when reconstructing 40 events forward, 21.1% for 100 events forward) and stays constant for $n \geq 30$, so for consistency with the experiments from Sucholutsky et al. [2019] we also use $n = 40$. Tables 3 and 6 which are described in the Results section offer some more benchmark results and some comparisons in accuracy between Restorer, LSTM, and this benchmark model.

3.3 Results

The models are trained using Adam optimizer [Kingma and Ba, 2014] with a fixed learning rate for 30 epochs on a randomly selected trace, before moving on to another randomly selected trace and repeating the procedure. Restorer is the restricted neighbourhood version of Transformer that was briefly mentioned as a future direction in Vaswani et al. [2017], so input sequence length can be limited to a neighbourhood of size 40 to be consistent with the LSTM model. As a result, each model received 40 consecutive messages as input and was asked to predict the next 40 messages as output. We trained both models using

Table 3. Maximum percent accuracy after n epochs when trained with input and output lengths of 40

Model							30-ep. acc	300-ep. acc	3000-ep. acc
Benchmark							30.07	30.07	30.07
LSTM	n_{hid}	n_{lstm}	d_h	d_{lstm}	n_{steps}				
	3	2	88	352	40		67.32	86.72	88.59
Restorer	N_{blocks}	d_{model}	d_k	d_v	d_h	n_{head}			
(main)	1	15	64	64	512	4	62.88	90.01	92.56
	2	15	64	64	512	4	74.76	88.12	93.41
	3	15	64	64	512	4	74.19	89.56	**94.00**
	5	15	64	64	512	4	74.17	83.01	84.44
	10	15	64	64	512	4	9.46	9.47	-
(1-head)	1	15	64	64	512	1	69.40	88.22	90.04
	2	15	64	64	512	1	**74.79**	89.87	92.71
	3	15	64	64	512	1	72.01	90.00	92.83
(mini)	1	15	32	32	88	1	68.91	87.01	88.85
	2	15	32	32	88	1	71.67	85.83	91.49
	3	15	32	32	88	1	71.66	89.85	93.15
(mini2)	1	32	16	16	32	1	71.66	91.02	93.28
	2	32	16	16	32	1	73.74	90.89	94.01
	3	32	16	16	32	1	74.60	**91.27**	93.69
(micro)	1	16	8	8	16	1	59.09	84.80	86.45
	2	16	8	8	16	1	64.87	85.35	90.33
	3	16	8	8	16	1	64.29	88.96	92.22

Result: Dictionary D containing all states and transition frequencies found in the training data
$i = 0$;
while $i + n < length(train\_data)$ **do**
    $K = train\_data[i : i + n]$;
    if $K \notin D.keys()$ **then**
        Let S_K be a new dictionary;
        for *each m_i in the set of the d unique messages* **do**
            $S_K[m_i] := 0$;
        end
        $D[K] := S_K$;
    end
    $m^* = train\_data[i + n]$;
    $S_K[m^*] = S_K[m^*] + 1$;
    $i = i + 1$;
end

Algorithm 1: Iteratively learn transition frequencies

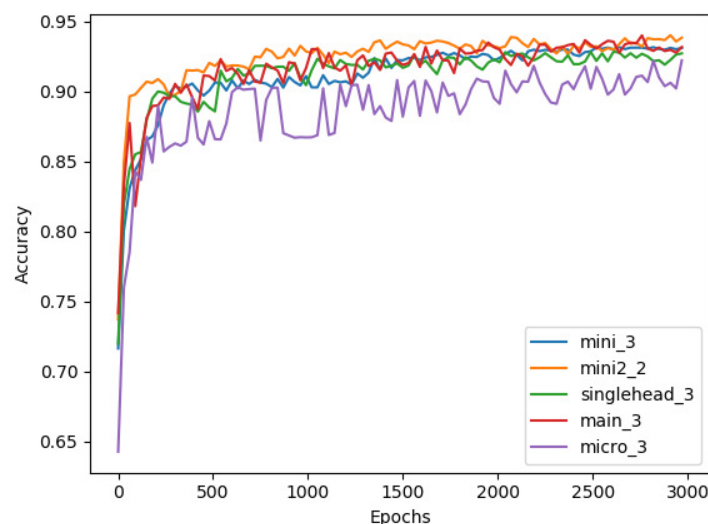
this training protocol, described in Sucholutsky et al. [2019], to compare the accuracy of the Restorer to that of the LSTM model in a consistent way. One important change when using the restricted version of Transformer is that the model no longer needs to learn to use an end of sequence token as length of the output is predetermined.

Table 3 illustrates that the majority of the different Restorer combinations outperform the LSTM model by several percent, and the benchmark by much more, when using the 40 previous events to predict the next 40. Interestingly, the best performance seems to be achieved by smaller architectures with either 2 or 3 identical blocks, although the smallest architecture, micro, has a noticeable drop in accuracy. The larger architectures seem to primarily suffer from overfitting, as training accuracy quickly begins to exceed

Table 4. Training time in seconds on single V100 GPU

Model	30-ep. time	300-ep. time
LSTM	77.64	807.25
Restorer (main, $N_{blocks} = 3$)	36.22	231.95

validation accuracy during their training. Models with 10 or more blocks did not converge at all past 10% test accuracy, likely due to the fixed learning rate being used. Figure 3 shows the full testing accuracy for the top configuration of each Restorer model version. While the increase in testing accuracy certainly slows down by 3000 epochs, we found that even after 6000 epochs small increases were still being made.

Figure 3. Testing accuracy of best Restorer configurations measured every 30 epochs of training. Model titles in the legend follow the format [model name]_[# of blocks].

On an Amazon Web Services (AWS) p3.2xlarge machine with a single V100 GPU and a batch size of 512, we found that the Restorer main model with 3 blocks was on average 2-4 times as fast as the LSTM model, as shown in Table 4.

While this method of training by selecting one random trace at a time does achieve high accuracy and lead to faster training times, we found that by concatenating all of the training data and training Restorer on it all at once, higher accuracy can be achieved. Table 5 shows that the Restorer achieves one to two percent higher accuracy when trained in this way instead of the trace-by-trace method.

We also found that when Restorer was trained exclusively with output lengths of 40, it generalized poorly to test cases with longer or shorter output sequences. In order to train models that would generalize to other output lengths, we found that changing the output length every 30 epochs to a random integer between 1 and 100 was an effective method. Table 6 summarizes the average accuracy achieved by Restorer models when tested with different output lengths. In this scenario, models with more blocks generalized better than those with fewer. It appears that when the length of the output sequence is fixed and known in advance, smaller models should be used, but when this length is either unknown or variable, it is better to use larger models and train in this stochastic way.

Table 5. Maximum percent accuracy after n epochs when trained with input and output lengths of 40 and using all training data at once

Model							30-ep. acc	300-ep. acc	3000-ep. acc
Restorer (main)	N_{blocks}	d_{model}	d_k	d_v	d_h	n_{head}			
	1	15	64	64	512	4	93.72	94.69	94.79
	2	15	64	64	512	4	**94.16**	**95.06**	**95.10**
	3	15	64	64	512	4	91.48	94.72	94.82
	5	15	64	64	512	4	84.36	84.43	93.11

Table 6. Percent accuracy on different output lengths after training using all data at once for 3000 epochs with input length of 40 and a new target output length randomly selected every 30 epochs

Model		1 out acc	10 out acc	20 out acc	40 out acc	60 out acc	80 out acc	100 out acc
Benchmark		76.55	45.41	37.01	30.07	25.70	22.99	21.10
Restorer (main)	N_{blocks}							
	1	91.22	92.09	92.23	92.16	75.21	56.41	45.13
	2	**91.98**	**92.49**	**92.54**	92.44	78.65	58.98	47.18
	3	91.89	92.37	92.44	**92.58**	**92.56**	**92.56**	**88.41**
	5	91.66	91.99	91.94	92.08	92.21	92.31	86.96

3.4 Case Study: Timed Regular Expression Mining on CAN Traces

It is important to verify that the quality of restoration is not just high in terms of accuracy, but also in terms of the effect on end-stage algorithms that would make use of the restored data. For consistency, we perform a case study using the same complex downstream algorithm as in Sucholutsky et al. [2019].

Time of occurrence of an event plays a key role in the domain of real-time systems [Lamport, 1978, Dwyer et al., 1999]. Recently, there has been tremendous activity in reverse engineering of complex real-time systems by mining temporal properties, typically by making use of template-based mining frameworks, that reflect the common behavior of these systems [Lemieux et al., 2015]. Since most programs in industry and elsewhere lack formal specifications, mined specifications play a key role in the life of software as they can be used for tasks like testing [Dallmeier et al., 2010] or verification [Kincaid and Podelski, 2015].

State-machine based approaches have become quite popular in the domain of both active and passive learning of system specifications. The behavior of a system is learned in the form of a state machine that can be employed for numerous tasks such as run-time monitoring, run-time verification, debugging etc. In the context of real-time systems, the focus of these temporal specifications is inclined towards a quantitative (actual duration of time between events) notion of time rather than qualitative (ordering of events) notion of time. For example, the response of an interrupt handler should be bound within its predefined time constraints. These time constraints are critical to the safe operation of real-time systems since a delayed response can lead to a faulty system operation.

System traces used for mining temporal specifications for a system are quite often lossy due to various software and hardware issues such as a buffer overflow or a memory leak. Therefore, recovery of data/traces will play a very important role in not only improving understanding of system behavior but also performing better fault diagnosis. Mining of Timed Regular Expressions (TREs) [Eugene Asarin, 2002] was proposed by Cutulenco et al. [2016], Narayan et al. [2018].The method proposed by Eugene Asarin [2002] to synthesize a timed automaton for a given TRE is the basis of the proposed approach. The timed automaton is then used as a checker to verify whether traces satisfy the corresponding TRE.

The mining algorithm is executed on three types of traces for the purpose of evaluation: a) Normal

244 traces, b) Lossy traces, and c) Restored traces [Sucholutsky et al., 2019]. The Normal traces in this
245 case correspond to the original traces obtained from a real vehicle as described in Section 3. We do not
246 have a large corpus of lossy CAN traces so instead lossy traces were generated from Normal traces by
247 introducing random loss of events. We introduced loss of events at five levels: 5%, 10%, 15%, 20%,
248 and 25%. Higher levels of loss are extremely unlikely to occur in a real vehicle aside from total failure
249 of multiple components and as such are not included; nonetheless, exploratory experiments suggested
250 that final results for higher levels of simulated loss were consistent with the pattern seen among these 5
251 selected levels. All losses were random to ensure that we are able to approximate a real-world scenario,
252 as bugs, sensor defects, buffer overflows, cyberattacks, disconnects, and other similar events can all cause
253 loss at any point within a trace. Lastly, the Restored traces were obtained from the Restorer main model
254 with $N_{blocks} = 3$ that was trained on the Normal traces. We ensure that timestamps remain in order with
255 the Normal traces since both time and order of occurrence of events is of key importance for specification
256 in the form of TREs. Two TRE templates from Narayan et al. [2018] are used with a time interval
257 parameter of 0 to 1,000 as was done in Sucholutsky et al. [2019].

258 **T-1(response)**: $(\langle P\rangle^*.(\langle P.\langle S\rangle^*.S\rangle[0,1000]).\langle P\rangle^*)+$

259 **T-2(alternating)**: $(\langle P|S\rangle^*.(\langle P.\langle P|S\rangle^*.S.\langle P|S\rangle^*\rangle[0,1000]))+$

Table 7. Percentage deviation in total number of mined TRE instances in restored traces and lossy traces at each level of loss when compared to the number mined in normal traces

Trace Type	Loss = 5%	Loss = 10%	Loss = 15%	Loss = 20%	Loss = 25%
Lossy Traces	6.2%	17.8%	21.6%	33.37%	53.99%
Restored Traces	1.16%	4.6%	6.55%	7.96%	9.53%

260 We evaluated the performance of the Restorer main model with $N_{blocks} = 3$ by comparing the percent
261 of TRE instances mined by the framework of Narayan et al. [2018]. In Table 7, we present the percent of
262 instances that were not found in the lossy and restored traces when compared with normal traces. Low
263 deviation values are desirable as deviation is indicative of information loss. It is evident from the results
264 in Table 7 that the quality of the Restorer restoration is extremely high. The Restorer is able to reduce the
265 deviation in number of mined instances from 6.2% to 1.16% in case of low levels of loss and there is a
266 remarkable reduction of almost 45% in the deviation of number of instances mined in cases with high
267 levels of loss.

4 CONCLUSION

269 Restoring lost data is an important intermediate problem in machine learning as many algorithms rely
270 on clean data for input. Previous solutions relied on LSTMs to perform the restoration of discrete data
271 in sequences. We have demonstrated that models making use of attention mechanisms based on the
272 recently proposed Transformer model are faster, smaller, and achieve better accuracy than the LSTM
273 restoration model. We have shown that the exact architecture can be tailored to the task at hand based on
274 the associated constraints. For example, for compute-constrained or time-constrained tasks, a smaller
275 architecture can be chosen to minimize training or inference times and number of parameters. Meanwhile,
276 for tasks where the length of the output sequence is variable or unknown beforehand, a larger architecture
277 with more blocks achieves better accuracy.

278 There are several limitations of this study that merit discussion. First off, we avoided using bidirec-
279 tional methods because they would have a large impact on the latency of algorithms running downstream
280 from Restorer when working with streams of data. However, in order to improve accuracy, it may be
281 useful to develop a mechanism for using newly processed steps of the input sequence to retroactively
282 correct predictions made by Restorer. Second, while we have tested Restorer on real data, it will be
283 important to conduct additional studies with other datasets to further establish Restorer as an effective
284 solution for data restoration. Finally, we have shown that Transformer-based models can be used for even
285 more sequence modelling tasks than the already wide-range described in Vaswani et al. [2017]; however,
286 more experiments with a larger variety of tasks may need to be conducted before Transformer-based
287 models are fully accepted as a state-of-the-art framework for general sequence modelling rather than just

288 natural language modelling. That being said, the results from the CAN data experiments and the case
289 study are certainly promising.
290   In general, we have shown that Restorer can achieve accuracies of up to 95% when restoring long
291 sequences of missing CAN data, beating out the benchmark method by a wide margin. When comparing
292 Restorer to LSTM-based methods for restoring CAN data, we demonstrated that using Restorer leads to a
293 reduction in model sizes by up to 2 orders of magnitude, an up to 4-fold reduction in training times, and
294 an increase of up to 7% in accuracy of data restoration. We additionally demonstrated in a case study that
295 our method successfully improves the performance of complex downstream algorithms.

REFERENCES

Abadi, M., Chu, A., Goodfellow, I., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016). Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM.

Aljuaid, T. and Sasi, S. (2016). Proper imputation techniques for missing values in data sets. In *International Conference on Data Science and Engineering (ICDSE), 2016*, pages 1–5. IEEE.

Altinel, F., Ozay, M., and Okatani, T. (2018). Deep structured energy-based image inpainting. In *2018 24th International Conference on Pattern Recognition (ICPR)*, pages 423–428. IEEE.

Andridge, R. R. and Little, R. J. (2010). A review of hot deck imputation for survey non-response. *International Statistical Review*, 78(1):40–64.

Beaulieu-Jones, B. K. and Moore, J. H. (2017). Missing data imputation in the electronic health record using deeply learned autoencoders. *Biocomputing*.

Blend, D. and Marwala, T. (2008). Comparison of data imputation techniques and their impact. *arXiv preprint arXiv:0812.1539*.

Buuren, S. v. and Groothuis-Oudshoorn, K. (2010). mice: Multivariate imputation by chained equations in r. *Journal of Statistical Software*, pages 1–68.

Chollet, F. (2017). Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1251–1258.

Cover, T. M. and Hart, P. E. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

Cutulenco, G., Joshi, Y., Narayan, A., and Fischmeister, S. (2016). Mining timed regular expressions from system traces. In *Proceedings of the 5th International Workshop on Software Mining*, SoftwareMining 2016, pages 3–10, New York, NY, USA. ACM.

Dallmeier, V., Knopp, N., Mallon, C., Hack, S., and Zeller, A. (2010). Generating test cases for specification mining. In *Proceedings of the 19th International Symposium on Software Testing and Analysis*, pages 85–96. ACM.

Donders, A. R. T., Van Der Heijden, G. J., Stijnen, T., and Moons, K. G. (2006). A gentle introduction to imputation of missing values. *Journal of Clinical Epidemiology*, 59(10):1087–1091.

Duan, Y., Lv, Y., Kang, W., and Zhao, Y. (2014). A deep learning based approach for traffic data imputation. *17th International IEEE Conference on Intelligent Transportation Systems (ITSC)*.

Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. (1999). Patterns in Property Specifications for Finite-State Verification. In *Software Engineering, 1999. Proceedings of the 1999 International Conference on*, pages 411–420. IEEE.

Efron, B. (1994). Missing data, imputation, and the bootstrap. *Journal of the American Statistical Association*, 89(426):463–475.

Eugene Asarin, Paul Caspi, O. M. (2002). Timed Regular Expressions. *J. ACM*, 49(2):172–206.

García, S., Luengo, J., and Herrera, F. (2015). *Data preprocessing in data mining*. Springer.

Gers, F. A. and Schmidhuber, E. (2001). LSTM recurrent networks learn simple context-free and context-sensitive languages. *IEEE Transactions on Neural Networks*, 12(6):1333–1340.

Glorot, X., Bordes, A., and Bengio, Y. (2011). Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 513–520.

Gondara, L. and Wang, K. (2017). Multiple imputation using deep denoising autoencoders. *CoRR*, abs/1705.02737.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680.

Hochreiter, S., Bengio, Y., and Frasconi, P. (2001). Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. In Kolen, J. and Kremer, S., editors, *Field Guide to Dynamical Recurrent Networks*. IEEE Press.

Huang, C.-Z. A., Vaswani, A., Uszkoreit, J., Shazeer, N., Hawthorne, C., Dai, A. M., Hoffman, M. D., and Eck, D. (2018). An improved relative self-attention mechanism for transformer with application to music generation. *arXiv preprint arXiv:1809.04281*.

Kalchbrenner, N., Espeholt, L., Simonyan, K., Oord, A. v. d., Graves, A., and Kavukcuoglu, K. (2016). Neural machine translation in linear time. *arXiv preprint arXiv:1610.10099*.

Kincaid, Z. and Podelski, A. (2015). Automated Program Verification. In *Language and Automata Theory and Applications: 9th International Conference, LATA 2015, Nice, France, March 2-6, 2015, Proceedings*, volume 8977, page 25. Springer.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.

Kotsiantis, S., Kanellopoulos, D., and Pintelas, P. (2006). Data preprocessing for supervised leaning. *International Journal of Computer Science*, 1(2):111–117.

Lakshminarayan, K., Harp, S. A., Goldman, R. P., and Samad, T. (1996). Imputation of missing data using machine learning techniques. In *KDD*, pages 140–145.

Lamport, L. (1978). Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565.

LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. *Nature*, 521(7553):436.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., and Wang, Z. (2017). Photo-realistic single image super-resolution using a generative adversarial network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4681–4690.

Lehtinen, J., Munkberg, J., Hasselgren, J., Laine, S., Karras, T., Aittala, M., and Aila, T. (2018). Noise2noise: Learning image restoration without clean data. *arXiv preprint arXiv:1803.04189*.

Leke, C., Marwala, T., and Paul, S. (2015). Proposition of a theoretical model for missing data imputation using deep learning and evolutionary algorithms. *CoRR*, abs/1512.01362.

Lemieux, C., Park, D., and Beschastnikh, I. (2015). General LTL Specification Mining. In *Automated Software Engineering (ASE), 2015 30th IEEE/ACM International Conference on*, pages 81–92. IEEE.

Lesher, G. W., Moulton, B. J., and Higginbotham, D. J. (1999). Effects of ngram order and training text size on word prediction. In *Proceedings of the RESNA'99 Annual Conference*, pages 52–54. Citeseer.

Lin, Y., Michel, J.-B., Aiden, E. L., Orwant, J., Brockman, W., and Petrov, S. (2012). Syntactic annotations for the google books ngram corpus. In *Proceedings of the ACL 2012 system demonstrations*, pages 169–174. Association for Computational Linguistics.

Lsdefine (2018). Lsdefine/attention-is-all-you-need-keras.

Luong, T., Pham, H., and Manning, C. D. (2015). Effective approaches to attention-based neural machine translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1412–1421.

Miller, J. and Hardt, M. (2018). When recurrent models don't need to be recurrent. *arXiv preprint arXiv:1805.10369*.

Myers, T. A. (2011). Goodbye, listwise deletion: Presenting hot deck imputation as an easy and effective tool for handling missing data. *Communication Methods and Measures*, 5(4):297–310.

Narayan, A., Cutulenco, G., Joshi, Y., and Fischmeister, S. (2018). Mining timed regular specifications from system traces. *ACM Trans. Embed. Comput. Syst.*, 17(2):46:1–46:21.

Niklaus, S., Mai, L., and Liu, F. (2017). Video frame interpolation via adaptive convolution. *CoRR*, abs/1703.07514.

Panzner, M. and Cimiano, P. (2016). Comparing hidden markov models and long short term memory neural networks for learning action representations. In *International Workshop on Machine Learning, Optimization and Big Data*, pages 94–105. Springer.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/languageunsupervised/language understanding paper. pdf*.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1:8.

Raghunathan, T. E., Lepkowski, J. M., Van Hoewyk, J., and Solenberger, P. (2001). A multivariate technique for multiply imputing missing values using a sequence of regression models. *Survey Methodology*, 27(1):85–96.

Rives, A., Goyal, S., Meier, J., Guo, D., Ott, M., Zitnick, C. L., Ma, J., and Fergus, R. (2019). Biological structure and function emerge from scaling unsupervised learning to 250 million protein sequences. *bioRxiv*.

Safavian, S. R. and Landgrebe, D. (1991). A survey of decision tree classifier methodology. *IEEE transactions on systems, man, and cybernetics*, 21(3):660–674.

Schafer, J. L. and Graham, J. W. (2002). Missing data: our view of the state of the art. *Psychological*

406   *Methods*, 7(2):147.

407   Schmitt, P., Mandel, J., and Guedj, M. (2015). A comparison of six methods for missing data imputation.
408   *Journal of Biometrics & Biostatistics*, 6(1):1.

409   Schonlau, M., Guenther, N., and Sucholutsky, I. (2017). Text mining with n-gram variables. *Stata Journal*,
410   17(4):866–881.

411   Sucholutsky, I., Narayan, A., Schonlau, M., and Fischmeister, S. (2019). Deep learning for system trace
412   restoration. In *2019 International Joint Conference on Neural Networks (IJCNN)*. IEEE. Pre-print at
413   arXiv:1904.05411.

414   Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks.
415   *CoRR*, abs/1409.3215.

416   Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin,
417   I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems*, pages
418   6000–6010.

419   Williams, R. J. and Zipser, D. (1989). A learning algorithm for continually running fully recurrent neural
420   networks. *Neural Computation*, 1(2):270–280.

421   Xie, J., Xu, L., and Chen, E. (2012). Image denoising and inpainting with deep neural networks. In
422   *Advances in neural information processing systems*, pages 341–349.

423   Zhou, J. and Huang, Z. (2017). Recover missing sensor data with iterative imputing network. *CoRR*,
424   abs/1711.07878.