Enhancing LSTM Models with Self-Attention and Stateful Training

Alexander Katrompas and Vangelis Metsis

Texas State University, San Marcos, TX 78666, USA
amk181@txstate.edu, vmetsis@txstate.edu

Abstract. When using LSTM networks to model time-series data, the standard approach is to segment the continuous data stream into fixed-size sequences and then independently feed each sequence to the LSTM network for training in a stateless fashion (i.e in a fashion that resets the LSTM cell state per fixed-size sequence). As a result, long-term dependencies between patterns appearing in the data stream may be lost. In this work, we introduce a hybrid deep learning architecture that enables long-term inter-sequence modeling while maintaining focus on each sequence's local characteristics. We use stateful LSTM training to model long-term dependencies that span the fixed-size sequences. We also utilize the attention mechanism to optimally learn each training sequence by focusing on the parts of each sequence that affect the classification outcome the most. Our experimental results show the advantages of each of these two mechanisms independently and in conjunction, compared to the standard stateless LSTM training approach.

Keywords: recurrent neural networks, lstm, deep learning, attention mechanisms, time series data, self-attention

1 Introduction

Recurrent neural networks (RNNs) are well known for their ability to model temporal dynamic data, especially in their ability to predict temporally correlated events [24]. RNNs form a family of neural networks in which a key feature is the additional input of the previous time-step's network "state," also known as "memory." This memory allows RNNs to retain temporal relationships by creating an association between the current time-step and the previous time-step, thereby representing a chain of causation [24, 8].

A vanilla RNN's memory length is relatively short and typically newer information is weighted heavier than older information. However, ideally, an RNN should not only retain longer past information, but it would also weigh information based on importance to the model and not simply on its recent proximity in time. Well established developments in these areas are the RNN architectural variant known as the Long Short-Term Memory (LSTM) network [9, 15], as well as the Back-propagation Through Time (BPTT) learning algorithm [28, 4]. In this study, LSTM networks and variants of BPTT will be studied with the further enhancements: attention mechanisms and stateful training. The goal of such

enhancements is enhancing RNN memory in memory length (stateful training), feature importance, and inter-sequence weighting (self-attention).

We have built a hybrid deep neural network architecture that enhances the ability of LSTM networks to both "focus" on importance within a sequence and "remember" long term patterns, thereby not only increasing accuracy but also reducing or eliminating the need for extensive data preparation. The first enhancement, a mechanism known as attention [1, 11], allows the network to focus on more salient sequences within the LSTM memory space. Specifically, in this discussion, we examine self-attention, also known as intra-attention, which focuses on the important relationships between features within sequences [25, 10]. This enhancement is network-level architectural in nature, altering the structure of the network while leaving the LSTM layer untouched. The second enhancement, a training model enhancement, overrides the typical LSTM back-propagation through time algorithm. This enhancement, which we term "stateful training," allows the LSTM layer to retain its state between error correction updates while also retaining its "batch update" behavior, thereby capturing long sequences of information in an efficient manner. These enhancements are studied individually and in conjunction so that four models are compared and contrasted for temporal classification performance: 1) baseline LSTM, 2) LSTM w/Attention, 3) stateful LSTM, 4) stateful LSTM w/Attention.

The remainder of this paper is organized as follows. We first introduce some background work on recurrent neural networks, LSTM networks, and the attention mechanism. Subsequently, we describe the details of our methodology and our proposed solution. We then evaluate our method and compare its performance against a baseline LSTM model as well as against results of other studies on the same publicly available datasets. We discuss our observations on the training behavior of the proposed architecture. Finally, we summarize and conclude this work.

2 Background

RNNs and their associated learning algorithms are typically some variation or enhancement to the standard feed-forward neural network architecture and the back-propagation learning algorithm. A brief introduction to the feed-forward back-propagation (FFBP) algorithm is presented here to frame the challenge and solutions presented in this study. More details about these algorithms can be found in [5, 22, 25].

2.1 Feed-Forward Networks, Recurrent Neural Networks, Back Propagation Through Time

A FFBP network trains very simply by feeding information through the network forward, and then back-propagating errors in the reverse, typically with some form of gradient descent. In the simplest case, each neuron's activation

in the network is "fed forward" through a simple sigmoid activation per neuron, errors are calculated in the output layer, and back-propagated through the network for correction of weights between neurons. This feed-forward and back-propagation process is executed with each iteration through the data. A complete pass through all data is known as an epoch [6].

A RNN and its training is derived from the basic FFBP network. The most basic RNN simply captures its current "state" as the output of the RNN layer, and "feeds" this output back to itself as an extra input in the next time step. Typically the RNN structure forms the first layer of a deeper network where a FFN is fed from the output of the RNN. Layered RNNs are also common. In the case of a layered network, the output of the RNN is "hidden" within the network and is therefore called a hidden layer, its neurons termed hidden nodes, and its output termed hidden output [21].

In addition to the aforementioned architectural change, the BP learning algorithm is typically modified into what is known as Back-propagation Through Time (BPTT). Rather than updating the weights with each iteration of input data, input is "batched," where each batch is some uniform fraction of the total data. Data are fed forward in batches without error correction, collecting all neural output, and updating the network over the entire batch at once [4, 23, 28].

2.2 Long Short-Term Memory and Truncated BPTT

Long Short-Term Memory (LSTM) networks are a variant of RNN which not only feed the previous hidden output back into the input of the LSTM but also maintain a separate "cell state," which updates with each iteration, independent of batch error correction. This cell state is not directly affected by the back-propagation of errors thereby giving the network the ability to avoid the well-known vanishing/exploding gradient problem [21]. Unlike vanilla RNNs, LSTMs can learn tasks which require memories of events that happened hundreds of discrete time-steps earlier [21, 9].

LSTMs also use the batched BPTT algorithm using (aka Truncated BPTT). In typical TBPTT some batch size, k, is chosen between 2 and $n/2$ where n is the number of instances in the training set. When training an LSTM the internal cell state of the LSTM is typically reset between batches. This reset effectively removes the ability of the network to retain state (i.e. memory) across batches. A form of TBPTT time that allows for information flow across boundaries is known as accelerated TBPTT (A-TBPTT). In this case, $k1$ is chosen to a batch size and $k2$ the error size, where $k1 < k2$. In other words, $k1$ is *when* to correct the network and $k2$ is the *amount* by which to correct. In this fashion, some portion of a previous batch's state is incorporated into the current batch [4, 23, 28].

Extending the idea of A-TBPTT it can be imagined the network could be trained using TBPTT but trained using maximal information on everything seen to that point (i.e. $k1 < k2$ where $k2$ is all information seen to that point). The advantage to this could be to both take advantage of TBPTT while also

maintaining maximal state information from one batch to the next. However, if it were simply a matter of choosing $k1$ normally, and $k2$ to be n (i.e. all instances), to retain all state information, this would simply devolve into a very inefficient form of classical BPTT (i.e. $k = n$). This method of training also tends to "overload" the network with long past information unlikely to be relevant to the current time-step thereby creating noise.

2.3 Self-Attention

Attention mechanisms are a well-known technique in natural language processing using Seq2seq encoder-decoder models. Standard encoder-decoders generally operate with the encoder processing the input sequence and then "compressing" or "summarizing" the information into a context vector of a fixed length for passing to the decoder. A disadvantage of this fixed-length context vector is the inability of the system to remember longer sequences as well as weighing recent information as more important regardless of its true relevance. Attention mechanisms are designed to resolve these problems. [1, 11]

Self-attention, also known as intra-attention, is an attention mechanism relating different positions of a sequence in order to model dependencies between different parts of the sequence. This differs from general attention in that instead of seeking to discover the "important" parts of the sequence relating to the network output, self-attention seeks to find the "important" portions of the sequence that relate to each other. This is done in order to leverage those intra-sequence relationships to improve network predictions. [10, 3, 16, 17, 25]

Originally designed for text processing, the benefit of self-attention can be seen in the following example. In order to understand the sentence, "*the dog did not run home because it was too tired*," the word "*it*" must be related to the word "*dog*" or the sentence makes no sense. However, if we change the word "*tired*" to "*far*," then the word "*it*" must be related to the word "*home*." Obviously, the relationship between "*it*" and the subject of the sentence is extremely important to the general understanding of the sentence as a whole. In general attention, the mechanism would seek to process the entire sentence and then emphasize the portions that are most important based on the correctness of network output. Conversely, self-attention seeks to relate portions of the sentence that are most important to each other prior to prediction, thereby enhancing understanding and prediction in a more context-specific manner. [10, 3, 16, 17] This technique has proven so successful that in the case of text processing it has been shown to stand alone without the need for RNN or CNN layers and perform as well or better on its own. [25]

2.4 Experimental Rationale

While it could be argued that text processing is temporal in nature since words have order and are related through time, strictly speaking, text processing is not time-series data. In fact, it could be argued that a text sentence, or even an entire paragraph, is more related to an image in that it represents a single "picture"

conceptually in the mind of the reader. In fact, attention mechanisms designed for text processing found almost immediate further success being adapted to image processing. This further emphasizes this "single concept" idea between image and text processing.[29] The analogy goes further in that attention in a sentence or paragraph is generally focusing on subject/verbs/adverbs for understanding, just as in an image attention is focusing on objects/actions/attributes.

This study seeks to conduct a preliminary analysis on attention's efficacy on true time-series data, specifically in temporal classification tasks. As detailed in the landmark paper, *Attention is all you need* [25], the temporal layer can be removed from text and image processing. However, we seek to understand attention's role when the temporal aspect of the data is its primary feature. To test this, we re-introduce the LSTM layer and study the interplay between LSTM and attention layers where the LSTM layer is responsible for temporal relationships and attention is responsible for relationships between features. We propose that there is a benefit to understanding the data both "vertically" (i.e. through time) and "horizontally" (i.e. feature to feature) when learning true time-series data. This study seeks to investigate this empirically prior to the next logical step: theoretical study (should it prove worthy empirically).

3 Methodology

This study seeks to investigate the following challenge: *How do we maintain maximum relevant temporal state information, without picking up noise and irrelevant information, without over-training, while also leveraging relevant feature importance?* In other words, how do we make the LSTM maximally "stateful" but have it pay "attention" to only relevant information? The solutions proposed here study both the concepts of statefulness to preserve information through batches, and the concept of "attention" to focus training on specific, short-term, feature-to-feature, high-value information.

3.1 Statefulness

In the context presented here, "statefulness" refers to the LSTM's ability to preserve its cell state through batches [14, 19]. Typically LSTMs are trained without any preservation of state between batches (i.e. $k1 = k2 < n$ and $n\%k1 = 0$). This can be partially solved through A-TBPTT. It should be noted that carrying state forward is not always desirable and this is highly data-dependent. Stateful training on data which has many short-term dependencies, and/or causation is a near-term event, and/or the data has clear and uniform temporal "sections," may actually be harmful to the model's performance. However, what is of concern in this study is data that is continuous with longer-term relevant knowledge throughout the data.

To achieve this we begin with setting the batch size to 1. This is a matter of the TensorFlow/Keras API used to model the data, and not part of the general algorithm. Setting batch size to 1 has the effect of making the training

sequence equal to 1. This normally would cause the loss of all LSTM cell state information since the LSTM cell state will be reset with every iteration. However, we will alter the LSTM behavior to maintain state between batches (i.e. do not reset the cell state) by setting "stateful" to true (again, this is a matter of the TensorFlow/Keras API used as a method to achieve our algorithmic goals).

In this programmatic form (batch size = 1, stateful = true), training is analogous to classical BP, however, we will also structure the data into time slices from 10s to 100s of steps (i.e. a "sequence"), thereby allowing TBPTT to be performed. LSTM cell state will be reset only at the end of an epoch, as opposed to at the end of a sequence, and multiple epochs will be presented. This can be seen in algorithm 1 where the difference between common LSTM batched training and "stateful" training is the placement of the step, "reset LSTM cell state." In typical LSTM training, this is performed automatically and immediately following the step, "execute TBPTT." The end result of this altered training algorithm is an LSTM network that will maintain cell state throughout an entire data set (i.e. epoch) while still training and correcting in batches according to TBPTT [9, 23, 28, 12, 14, 19].

Algorithm 1: Stateful Training Algorithm

```
Data: 3D matrix of r (x) c (x) s, where r is the number of training
      instances per sequence, c is the number of features, s is the
      number of sequences, and where N%s=0, where N = total
      number training instances.
Initialize network;
while epochs remaining do
    foreach s do
        for i ← 1 to r do
            propagate s_i forward;
            E += e_i;
        end
        execute TBPTT;
    end
    reset LSTM cell state;
end
```

3.2 LSTM and Attention

When combined with LSTM architectures, attention operates by capturing all LSTM output within a sequence and training a separate layer to "pay attention" to (i.e. to weigh) some parts of the output more than others. Note that the LSTM is set to return sequences, i.e. for an input sequence $\boldsymbol{x} = (x_1, x_2, ..., x_T)$ the LSTM layer produces the hidden vector sequence $\boldsymbol{h} = (h_1, h_2, ..., h_T)$ and output $\boldsymbol{y} = (y_1, y_2, ..., y_T)$ of the same length, by iterating the following equations from $t = 1$ to T.

$$h_t = \mathcal{H}(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{1}$$

$$y_t = W_{hy}h_t + b_y \tag{2}$$

where the W terms denote weight matrices, the b terms denote bias vectors, and $\mathcal{H}$ is the hidden layer function. Details about LSTM networks can be found in [7].

Attention is essentially a *neural network within a neural network*, which is learning to weigh portions of a sequence for relative feature importance [27, 30]. The general concept of attention can be modified to work with temporal classification problems where the sequences are a collection of instances of time-series data and the "decoding" is classification. In the models presented here, rather than a sequence of words, the sequences are fixed-length vectors generated by segmenting the continuous data stream. Each value of the sequence vector is a time-step (data point) represented as a numeric value. This value can be a sensor measurement, a stock market price, etc. [18]. The attention used in this study is multiplicative self-attention[1] and uses the following attention mechanism:

$$h_t = tanh(W_x x_t + W_h h_{t-1} + b_h) \tag{3}$$

$$e_t = \sigma(x_t^T W_a x_{t-1} + b_t) \tag{4}$$

$$a_t = softmax(e_t) \tag{5}$$

where h_t is the hidden node output from the LSTM layer in a two-dimensional matrix (i.e. the entire hidden output achieved in Keras through setting *return sequences* to true). e_t is the sigmoid activation output of the attention two-layer network, where W_a is the attention network weights, producing a corresponding matrix of the attention network activations. a_t is the softmax activation of e_t producing a vector "alignment score" weighting the importance of the individual parts of the batched input sequences.

4 Data

In this section, we discuss the characteristics of the data for which the proposed architecture is advantageous as well as the datasets used in our experiments.

4.1 Data Characteristics

Sequential nature: The data to be modeled must be time-series data, continuous and in-order, sampled at reasonably regular rates, with dependencies through time. For example, environmental data such as barometric pressure, air moisture, current temperature, etc. in the prediction of future temperature. Gathering data such as environmental data, process control data, physio-metric data, biometric

[1] pypi.org/project/keras-self-attention/

data, etc. can be done continuously, in order, and at regular intervals, and is of high value to many classification problems.

Natural order: The data to be modeled must be reasonably natural and not artificially staged into discrete, disparate groups. For example, the data cannot be EEG data in ordered experimental events such as hearing a noise on the left/right, or a vision event on the left/right [20]. Since the events (auditory or visual stimulus) in this dataset follow a predetermined pattern scripted by the researchers, the model very quickly learns the experimental design pattern and not the EEG signal characteristics that are associated with the stimulus type. This leads to dramatic over-fitting and no generalizability. It should be noted this does not apply to data collected experimentally in which purposeful natural randomness is simulated with uneven events.

Temporal event classification: The classifications to be modeled must be temporal events through time, and not single-point, discrete classifications. In other words, the events being predicted are things that happen over time continuously. For example, predicting a human fall based on smart-device accelerometer readings. The movements leading up to a fall can be running, walking, standing, etc., followed by a fall which happens over time with a series of time-steps including the initial falling period, striking the ground, remaining in the fall position, recovery, and then back to some non-falling activity.

4.2 Data Sets

Three different datasets were used in our experiments.

SmartFall: The data set consists of raw (x, y, z) accelerometer readings representing activities of daily living (ADLs) such as walking and running with falls interspersed. [13]

MobiAct: The data set consists of raw (x, y, z) accelerometer readings with various ADLs (jogging, walking upstairs, falls, etc) recorded and labeled. [26]

Occupancy Data: The data set consists of recorded ambient features of an enclosed space (temperature, humidity, light, CO2, and humidity ratio) and the associated event label that space is occupied or not occupied for some period of time. [2]

In our experiments, the SmartFall and MobiAct data are not pre-processed other than to concatenate various subjects together into a single training, test, and validation set. Conversely, the original SmartFall study, and especially the MobiAct study, both do extensive pre-processing and feature extraction.

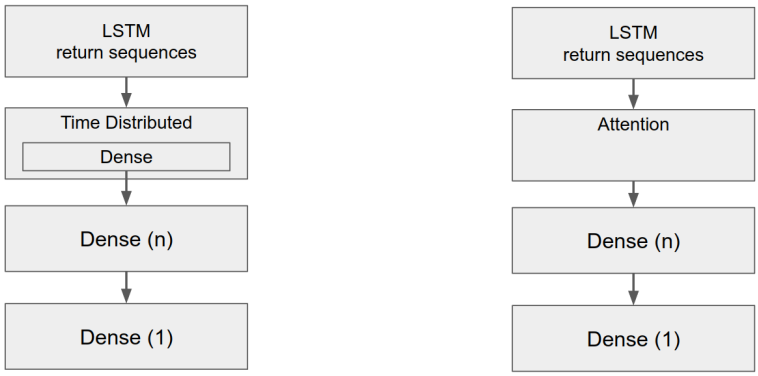
The occupancy data is not pre-processed and is taken as-is, in temporal order, in both our study and the cited work. However, the cited study does extensive statistical analysis to achieve the optimal model and feature set whereas our technique simply uses the data as-is with the complete feature set.

5 Models

Four models were used to demonstrate the effectiveness of the enhancements discussed here. All models are built using TensorFlow 2.0 with Keras and the third-party library mentioned above for achieving attention models.

5.1 Architectures

Model 1: Vanilla LSTM: This model is a typical LSTM deep-learning model and consists of an LSTM input layer, a dense layer wrapped in a time-distributed layer, another dense layer, and an output classifier. The LSTM return sequences parameter is set to true which enables the complete LSTM hidden layer sequences to be sent forward to the time distributed later as shown in Figure 1a. The time-distributed wrapper allows each set of hidden layer sequences to be applied to individual identical copies of the first dense layer. This conforms to the idea we want to capture and train on all hidden states equally, and not on just the resulting context vector of the hidden states. This also is analogous to the next model 1bwherein 'return sequences' is required to implement the attention layer. This also allows for a consistent comparison between models. The output of the time-distributed dense layer is forwarded to the subsequent dense layer, and finally to the output layer. It is assumed the reader is familiar with such models [9].



(a) Vanilla LSTM model. (b) LSTM with Attention.

Fig. 1: The figure shows the architectures of two networks designed for sequence classification.

Model 2: LSTM with Attention: This model replaces the time-distributed dense layer with an attention layer. Return sequences is set to true enabling the complete hidden layer sequences to be sent forward to the attention layer where they are processed similarly to the previously explained encoder/decoder model and the vanilla LSTM model (see Figure 1b).

Model 3: Stateful LSTM: This model is architecturally identical to the vanilla LSTM (Figure 1a), however, the learning algorithm is altered to maintain state as described in the section on stateful training. Both *return sequences* and *maintain state* parameters are set to true. The state is reset at the end of each epoch as described in algorithm 1.

Model 4: Stateful LSTM with Attention: This model utilizes the TensorFlow functional API and uses both stateful training and attention in parallel layers, which are then merged and fed forward to a common dense layer. In this model, each "side" of the network is trained according to its architecture as described in the previous two models respectively. (Figure 2).

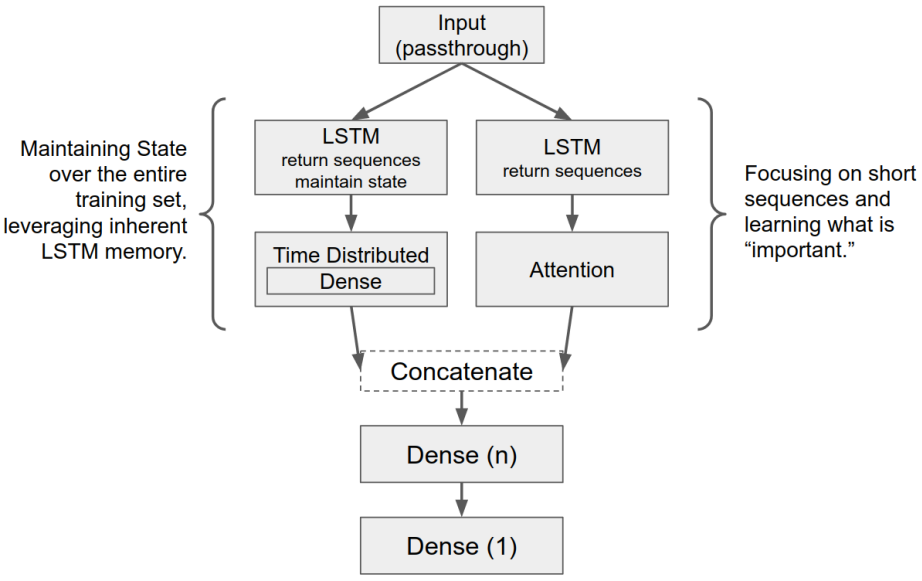


Fig. 2: Stateful LSTM with Attention

5.2 Hyperparameters

In each case, the models were tuned with the number of nodes, time-steps, and epochs that performed the best for the dataset at hand, so that the best performance of each was measured both against each other and against the existing published work. These parameters were selected in a grid search pattern varying hidden layer nodes, time-steps, and the number of epochs in all combinations until the optimal parameters were discovered for each model. Figure 3 shows the typical Stateful LSTM w/Attention summary. From this summary and the following general parameter ranges, it should be sufficient to reproduce all models.

Hidden Layer nodes were selected between 100 and 300 with an increasing number needed from models 1 to 4, in order, as described in the architectures sub-section.

Time-steps were chosen to be 40 in the case of an attention model and 200 in the case of a non-attention model (models 2 and 4, as described in the architectures sub-section).

Epochs were chosen between 120 and 35 in generally decreasing numbers from models 1 to 4, as described in the architectures sub-section. This is especially notable in that as the number of nodes increased from model to model, epochs decreased dramatically.

```
ATTENTION + STATEFUL MODEL
==========================
_________________________________________________________________________________
Layer (type)                    Output Shape   Param #  Connected to
=================================================================================
input_1 (InputLayer)            (1, 40, 5)     0
_________________________________________________________________________________
lstm_1 (LSTM)                   (1, 40, 256)   268288   input_1[0][0]
_________________________________________________________________________________
lstm_2 (LSTM)                   (1, 40, 256)   268288   input_1[0][0]
_________________________________________________________________________________
Attention (SeqSelfAttention)    (1, 40, 256)   65537    lstm_1[0][0]
_________________________________________________________________________________
time_distributed_1 (TimeDistrib (1, 40, 128)   32896    lstm_2[0][0]
_________________________________________________________________________________
concatenate_1 (Concatenate)     (1, 40, 384)   0        Attention[0][0]
                                                        time_distributed_1[0][0]
_________________________________________________________________________________
dense_1 (Dense)                 (1, 40, 128)   49280    concatenate_1[0][0]
_________________________________________________________________________________
dense_2 (Dense)                 (1, 40, 1)     129      dense_1[0][0]
=================================================================================
Total params: 684,418
Trainable params: 684,418
Non-trainable params: 0
Epochs: 35
```

Fig. 3: Typical Stateful LSTM with Attention Model used in the study.

6 Experiments and Results

We first present the experimental results of comparing the four different architectures studied in this work (i.e. Vanilla LSTM, LSTM w/ Attention, Stateful LSTM, and Stateful LSTM w/ Attention) against each other. We show these results per data set, including accuracy, precision, recall, and F1 scores. Figure 4 shows the bar graph of accuracy per data set. Finally, we compare the results of our best model (Stateful LSTM w/ Attention) with the results obtained by previously published work on the same datasets.

6.1 Model-to-Model and Model-to-Study Comparisons

Each of the tables 1 through 7 show the results of optimally training each model on each dataset. Tables 8 - 12 compare the results of each of the best models studied here (measured by accuracy) with the results from the cited studies from which each data set was acquired.

Table 1: SmartFall Fall Detection Results

SmartFall				
	LSTM	Attn	State	Attn State
Accuracy	.939	.946	.958	**.960**
Precision	.687	.777	.828	**.857**
Recall	.824	.809	.844	**.847**
F1	.750	.793	.836	**.852**
ROC AUC	.912	.941	.963	**.974**
PR AUC	.819	.827	.859	**.893**

Table 2: MobiAct: Fall Detection Results

MobiAct - Fall				
	LSTM	Attn	State	Attn State
Accuracy	.929	.936	.945	**.952**
Precision	.814	.799	.929	**.941**
Recall	.871	**.912**	.847	.864
F1	.841	.852	.886	**.901**
ROC AUC	.960	.966	**.990**	**.990**
PR AUC	.966	.933	.960	**.970**

Table 3: MobiAct: Jogging Detection Results

MobiAct - Jogging				
	LSTM	Attn	State	Attn State
Accuracy	.963	.970	.970	**.972**
Precision	.991	**.990**	**.990**	.988
Recall	.969	.977	.978	**.981**
F1	.980	.984	.984	**.985**
ROC AUC	.973	.980	**.982**	.965
PR AUC	.986	.996	**.997**	.991

Table 4: MobiAct: Detecting Walking Down Stairs

MobiAct - Stairs Down				
	LSTM	Attn	State	Attn State
Accuracy	.919	.943	.941	**.948**
Precision	.949	.960	.967	**.969**
Recall	.929	**.953**	.944	**.953**
F1	.939	.957	.955	**.961**
ROC AUC	.950	.965	**.968**	**.968**
PR AUC	.956	.955	.965	**.968**

Table 5: MobiAct: Detecting Walking Up Stairs

MobiAct - Stairs Up				
	LSTM	Attn	State	Attn State
Accuracy	.900	.919	.926	**.933**
Precision	.944	.953	.973	**.975**
Recall	.919	**.935**	.928	**.935**
F1	.931	.944	.950	**.955**
ROC AUC	.946	**.980**	.964	.973
PR AUC	.976	**.996**	.979	.989

Table 6: Detecting Occupancy of an Enclosed Space - Door Closed

Occupancy 1				
	LSTM	Attn	State	Attn State
Accuracy	.978	.961	.978	**.980**
Precision	.998	.996	.998	**.999**
Recall	.944	.903	.942	**.948**
F1	.907	.947	.969	**.973**
ROC AUC	.990	.991	**.994**	.990
PR AUC	.977	.980	.986	**.990**

Table 7: Detecting Occupancy of an Enclosed Space - Door Open

Occupancy 2				
	LSTM	Attn	State	Attn State
Accuracy	.925	.955	.948	**.970**
Precision	.778	.957	.922	**.993**
Recall	.860	.850	.840	**.890**
F1	.817	.901	**.879**	.939
ROC AUC	.984	**.993**	.984	**.993**
PR AUC	.929	.965	.959	**.972**

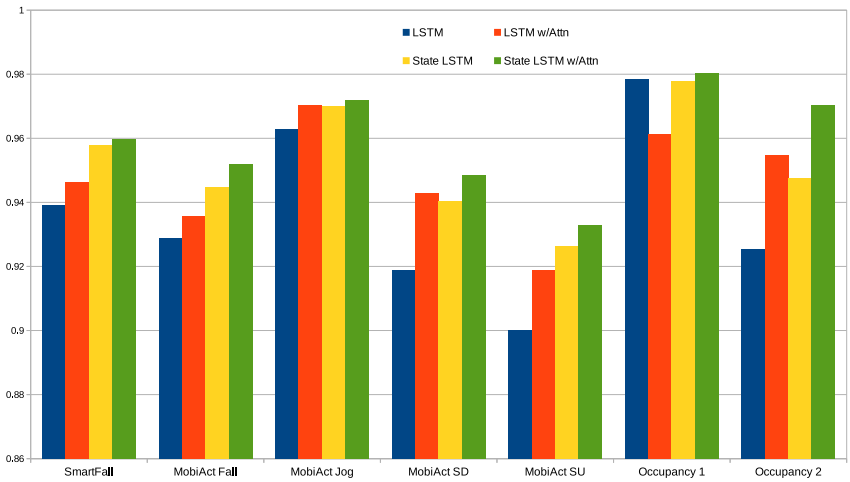


Fig. 4: Model to Model Accuracy Comparison

The first model-to-study comparison presented is the SmartFall study which used a combination of (x, y, z) accelerometer readings, derived features, and post-processing labels into categories of events as *fall* or *not fall*. Table 8 shows our results using only raw (x, y, z) accelerometer readings as input and no post-processing. Table 9 shows our results when post-processing is applied similar to the SmartFall study. Both results are compared to the deep learning model presented in the SmartFall study.

Also presented is the MobiAct fall detection results from our experiments, however, the MobiAct study did not include fall detection. The results of our MobiAct fall detection experiments are presented in table 10 as a comparison to both our SmartFall results and the original SmartFall study results. This is presented simply as a re-enforcement of the overall results of our models in a similar activity, with a different but comparable data set. Again, no pre-processing of our data was done, and we use post processing similar to SmartFall for comparison.

Table 11 shows the MobiAct results for jogging detection, walking downstairs, and walking upstairs. In each case, the input data for our models was raw accelerometer readings. Conversely, the input to the "multilayer perceptron" in the MobiAct study was a series of complex derived features that was termed in the study the *Optimal Feature Set* (OFS). This is notable in that we achieve results that in two out of three cases are superior. In the third case, our results have a lower accuracy score but are comparably close and notable given the difference in pre-processing effort.

Table 8: SmartFall: Stateful LSTM w/Attn versus Study without Post Processing

SmartFall w/o Post Processing		
	Attn State	Study
Accuracy	**.960**	.850
Precision	**.857**	.770
Recall	.847	**1.0**

Table 9: SmartFall: Stateful LSTM w/Att versus Study with Post Processing

SmartFall w/Post Processing		
	Attn State	Study
Accuracy	**.995**	.850
Precision	**1.00**	.770
Recall	.989	**1.0**

Table 10: Stateful LSTM w/Att SmartFall, Stateful LSTM w/Att MobiAct, SmartFall Comparision

MobiAct Fall Data versus SmartFall			
	Attn State Smart-Fall	Attn State Mobi-Act	Smart-Fall Study
Accuracy	**.995**	.984	.850
Precision	**1.0**	.968	.770
Recall	.989	**1.0**	**1.0**

Table 11: MobiAct Detecting ADLs versus Study

MobiAct Data						
	Jogging		Stairs-D		Stairs-U	
	Attn+ State	Study	Attn+ State	Study	Attn+ State	Study
Accy.	.972	**.996**	**.948**	.915	**.933**	.925

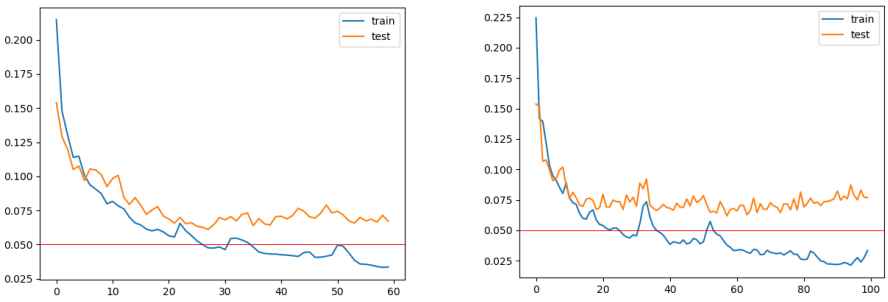
Table 12: Occupancy Detection versus Study

Occupancy Detection Data				
	Test 1		Test 2	
	Attn+ State	Study	Attn+ State	Study
Accy.	**.980**	.979	.970	**.993**

Table 12 shows occupancy detection compared to the cited study. This comparison is notable in that the cited study did not use a neural network model, but rather used several statistical models. Our results are presented as a comparison to these statistical models, specifically the best of statistical model results (Linear Discriminant Analysis). While our best results were similar to the cited study's best results, there are several things of note that make our approach novel and valuable. Again, our models used no pre-processing or pre-selection of inputs, whereas the cited study was in fact a study of the statistically "best" input selection. In other words, we achieved slightly better results (test set 1), and slightly worse but comparable results (test set 2), by simply using the entire feature set without the need for extensive comparative statistics. This comparison is of value as a demonstration that our enhanced deep learning models achieve similar or better results than most of the statistical methods in the cited paper.

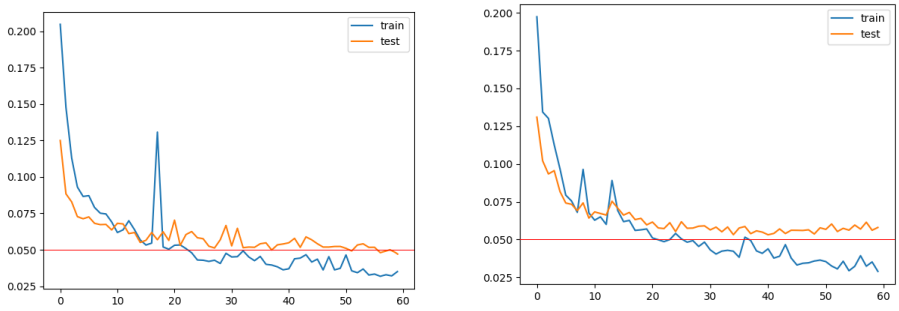
7 Discussion: Training Behavior

A notable and surprising effect on training became evident as the attention models were trained and studied. In each case, models that included attention resisted over-fitting, sometimes dramatically so. Even when trained well past the minimum achievable test error, the models did not exhibit over-fitting. This occurred in both the attention-only model and the stateful model with attention. Figures 5 through 7 show this effect. Figure 5 shows that as the standard LSTM model continued to train, the over-training effect becomes more and more pronounced. This was also observed in the stateful-only model (Figure 6b). However, in the attention models (Figure 6a and Figure 7) the test error closely parallels the training error as the training error continues to decline. Even when both errors "flat-lined," training and test error remained closer and parallel.



(a) 60 epochs (b) 100 epochs

Fig. 5: MobiAct Walking Down Stairs - Standard LSTM



(a) LSTM w/Attn (b) Stateful LSTM

Fig. 6: MobiAct Walking Down Stairs - Attention versus Statefulness

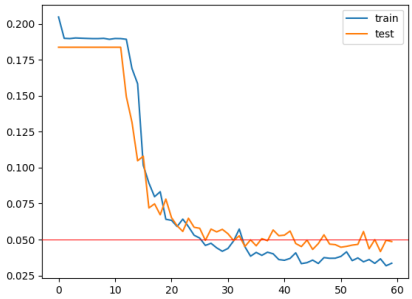


Fig. 7: MobiAct Walking Down Stairs - Stateful LSTM with Attention

This was an unexpected result, however, upon closer inspection, it is seemingly intuitive. The purpose of attention mechanisms is to reduce noise (i.e. irrelevant information) and focus on the relative "important" part of the sequences. It seems intuitive this should reduce over-fitting in that the model has a more difficult time memorizing the training set, and is instead constantly corrected to the important and predictive input sequences and feature relationships. However, this is only a preliminary hypothesis and necessitates further study and validation.

8 Conclusions

Our study conducted into LSTM model enhancements demonstrates clearly that LSTM models with the enhancements of statefulness and attention are capable of equal or better classification results than many state-of-the-art models, and most notably with far less pre-processing. This is an important finding in that pre-processing is not only cumbersome, it very often leads to human bias. With the enhancements presented here it is possible to effectively process raw data

into accurate temporal classification models. This is an important consideration when attempting to train models in real-time and online while the models are in service, an area that warrants further study.

In addition, this study demonstrates both the benefits of attention mechanisms as applied outside their typical domains (e.g. seq2seq text processing models) and re-examines the usefulness of a RNN layer(s) used in conjunction with attention for temporal classification. Furthermore, stateful training is an area gaining ground in the study of long-term pattern recognition and these results support those efforts.

Based on these results, attention mechanisms, specifically self-attention, can benefit from RNNs and vice versa, and that this is an area worthy of further investigation.

References

1. Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. arXiv preprint arXiv:1409.0473 (2014)
2. Candanedo Ibarra, L., Feldheim, V.: Accurate occupancy detection of an office room from light, temperature, humidity and co2 measurements using statistical learning models. Energy and Buildings 112 (12 2015)
3. Cheng, J., Dong, L., Lapata, M.: Long short-term memory-networks for machine reading (2016)
4. De Jeses, O., Hagan, M.T.: Backpropagation through time for a general class of recurrent network. In: IJCNN'01. International Joint Conference on Neural Networks. Proceedings (Cat. No.01CH37222). vol. 4, pp. 2638–2643 vol.4 (2001)
5. Dematos, G., Boyd, M.S., Kermanshahi, B., Kohzadi, N., Kaastra, I.: Feedforward versus recurrent neural networks for forecasting monthly japanese yen exchange rates. Financial Engineering and the Japanese Markets 3, 59–75 (1996)
6. Gershenson, C.: Artificial neural networks for beginners (2003)
7. Graves, A., Jaitly, N., Mohamed, A.r.: Hybrid speech recognition with deep bidirectional lstm. In: 2013 IEEE workshop on automatic speech recognition and understanding. pp. 273–278. IEEE (2013)
8. Hewamalage, H., Bergmeir, C., Bandara, K.: Recurrent neural networks for time series forecasting: Current status and future directions. International Journal of Forecasting (Aug 2020)
9. Hochreiter, S., Schmidhuber, J.: Long short-term memory. Neural computation 9, 1735–80 (12 1997)
10. Lin, Z., Feng, M., dos Santos, C.N., Yu, M., Xiang, B., Zhou, B., Bengio, Y.: A structured self-attentive sentence embedding (2017)
11. Luong, M.T., Pham, H., Manning, C.D.: Effective approaches to attention-based neural machine translation (2015)
12. Masters, D., Luschi, C.: Revisiting small batch training for deep neural networks (2018)
13. Mauldin, T., Canby, M., Metsis, V., Ngu, A., Rivera, C.: Smartfall: A smartwatch-based fall detection system using deep learning. Sensors 18(10), 3363 (Oct 2018)

14. Mohajerin, N., Waslander, S.L.: State initialization for recurrent neural network modeling of time-series data. In: 2017 International Joint Conference on Neural Networks (IJCNN). pp. 2330–2337 (2017)
15. Moldovan, D., Anghel, I., Cioara, T., Salomie, I.: Time series features extraction versus lstm for manufacturing processes performance prediction. In: 2019 International Conference on Speech Technology and Human-Computer Dialogue (SpeD). pp. 1–10 (2019)
16. Parikh, A.P., Täckström, O., Das, D., Uszkoreit, J.: A decomposable attention model for natural language inference (2016)
17. Paulus, R., Xiong, C., Socher, R.: A deep reinforced model for abstractive summarization (2017)
18. Qin, Y., Song, D., Chen, H., Cheng, W., Jiang, G., Cottrell, G.: A dual-stage attention-based recurrent neural network for time series prediction (2017)
19. Rahman, L., Mohammed, N., Al Azad, A.K.: A new lstm model by introducing biological cell state. In: 2016 3rd International Conference on Electrical Engineering and Information Communication Technology (ICEEICT). pp. 1–6 (2016)
20. Rivet*, B., Souloumiac, A., Attina, V., Gibert, G.: xdawn algorithm to enhance evoked potentials: Application to brain–computer interface. IEEE Transactions on Biomedical Engineering 56(8), 2035–2043 (2009)
21. Squartini, S., Paolinelli, S., Piazza, F.: Comparing different recurrent neural architectures on a specific task from vanishing gradient effect perspective. In: 2006 IEEE International Conference on Networking, Sensing and Control. pp. 380–385 (2006)
22. Struye, J., Latré, S.: Hierarchical temporal memory and recurrent neural networks for time series prediction:an empirical validation and reduction to multilayer perceptrons. Neurocomputing (04 2019)
23. Tang, H., Glass, J.: On training recurrent networks with truncated backpropagation through time in speech recognition (2018)
24. Tomiyama, S., Kitada, S., Tamura, H.: On a new recurrent neural network and learning algorithm using time series and steady-state characteristic. In: IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028). vol. 1, pp. 478–483 vol.1 (1999)
25. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is all you need (2017)
26. Vavoulas., G., Chatzaki., C., Malliotakis., T., Pediaditis., M., Tsiknakis., M.: The mobiact dataset: Recognition of activities of daily living using smartphones. In: Proceedings of the International Conference on Information and Communication Technologies for Ageing Well and e-Health - Volume 1: ICT4AWE, (ICT4AGEINGWELL 2016). pp. 143–151. INSTICC, SciTePress (2016)
27. Wang, Y., Huang, M., Zhu, X., Zhao, L.: Attention-based LSTM for aspect-level sentiment classification. In: Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing. pp. 606–615. Association for Computational Linguistics, Austin, Texas (Nov 2016), https://www.aclweb.org/anthology/D16-1058
28. Werbos, P.J.: Backpropagation through time: what it does and how to do it. Proceedings of the IEEE 78(10), 1550–1560 (1990)
29. Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhutdinov, R., Zemel, R., Bengio, Y.: Show, attend and tell: Neural image caption generation with visual attention (2016)
30. Zeng, J., Ma, X., Zhou, K.: Enhancing attention-based lstm with position context for aspect-level sentiment classification. IEEE Access 7, 20462–20471 (2019)