Long Short-term Memory Recurrent Neural Networks for Classification of Acute Hypotensive Episodes

by

Alexander Scott Jaffe

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2017

Author .....
Department of Electrical Engineering and Computer Science
May 26, 2017

Certified by .....
Una-May O'Reilly
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

Long Short-term Memory Recurrent Neural Networks for Classification of Acute Hypotensive Episodes

by

Alexander Scott Jaffe

Submitted to the Department of Electrical Engineering and Computer Science
on May 26, 2017, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

An acute hypotensive episode (AHE) is a life-threatening condition durich which a patient's mean arterial blood pressure drops below 60 mmHG for a period of 30 minutes. This thesis presents the development and evaluation of a series of Long short-term memory recurrent neural network (LSTM RNN) models which predict whether a patient will experience an AHE or not based on a time series of mean arterial blood pressure (ABP). A 2-layer, 128-hidden unit LSTM RNN trained with rmsprop and dropout regularization achieves sensitivity of 78% and specificity of 98%.

Thesis Supervisor: Una-May O'Reilly
Title: Principal Research Scientist

Acknowledgments

I immensely thank Una-May O'Reilly for her support throughout this project. Her guidance has led me to achieve something I for a while was unsure I would be able to complete and I have learned and gained much in the process.

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Motivation

Of the many sections in a hospital, none is more prone to life-threatening situations than the intensive care unit (ICU). The ICU provides care for patients who are seriously ill, and thus prone to experience a life-threatening episode at any moment. Among the many dangerous events that a patient can undergo in the ICU is the acute hypotensive episode (AHE). During an AHE, a patient's blood pressure drops perilously low, leading to a state of hypotension. AHEs are very dangerous for the patient since extended periods of very low blood pressure can deprive organs of oxygen, leading to shock and potentially death. It is thus vital for a patient who is at risk of experiencing an AHE to be identified as soon as possible, ideally even before the event occurs, and receive immediate care.

1.1 The PhysioNet AHE Prediction Challenge

The problem of predicting AHEs given ABP arose in part due to a challenge extended by PhysioNet and Computing in Cardiology in 2009. Participants were tasked with using various physiological data from the Multiparameter Intelligent Monitoring in Intensive Care (MIMIC-II) research database to accurately predict AHEs. For the purposes of this challenge, PhysioNet defined an AHE to be any 30 minute interval in which over 90% of the mean arterial pressure (MAP) measurements were at or below

60 mmHg. Also note that MAP is defined as

$$MAP = \frac{SBP + 2DBP}{3}. \tag{1.1}$$

SBP is systolic blood pressure and DBP is diastolic.

On account of this challenge, there have since been a variety of approaches to the prediction task, including but not limited to k-nearest-neighbor classification [10], latent state models [21], and even a simple threshold strategy [15]. We seek to further extend this body of work by applying a novel long short-term memory recurrent neural network model to the problem of predicting AHE.

1.2 Recurrent Neural Networks

In recent years, deep learning has transformed the field of machine learning. One of the neural network architecture paradigms that has driven breakthroughs in this field is recurrent neural networks (RNNs). RNNs are a kind of neural network that takes sequential input and produces sequential output by sharing parameters between time steps. RNNs have led to breakthrough results in natural language processing [20], image captioning [13], and speech recognition [7]. Though RNNs have proven useful in many sequence and series-based learning tasks, their application to raw time series prediction is still relatively unexplored. This thesis thus applies established neural network techniques to the established AHE prediction problem to produce novel results.

Chapter 2

Prior Work

This section provides a brief overview of preceding work specific to predicting AHE, and in the application of RNNs to physiologically-based time series.

2.1 Predicting AHEs

There is a wide body of work concerning the prediction of AHE, in no small part due to the PhysioNet challenge. Mneimneh et al. [15] explored three approaches to the problem: reconstructed phase-space neural networks, k-nearest neighbor, and a rule-based approach. The rule-based approach examined 20-minute intervals of MAP data with a lead time of 30 minutes. If the patient's mean MAP over the interval was less than 71 mmHg, the model would predict an AHE. This model performed surprisingly well with such a simple heuristic, achieving 92.85% sensitivity and 88.46% specificity. On account of this paper's results, we thus decided to use mean ABP as our primary input for the prediction task. Still, this result was found via direct observation of the dataset. We provide a more robust and less supervised approach.

Waldin et al. [21] employed latent state space models and used features such as the mean, standard deviation, and skew of ABP data over various time spans to try to infer hidden states. Dernoncourt et al.[3] decomposed ABP data into features via a wavelet transform learned via a gaussian process. Kim et al. revisited the k-nearest-neighbors approach, classifying the time series based on whether a majority of its

k-nearest-neighbors were AHE-positive or not [10]. This required a large database and the use of locality-sensitive hashing to reduce the resulting data structure to a tenable size. Though there exists previous work employing other kinds of neural networks to the AHE prediction problem [8], our work is the first to use RNNs.

2.2 Prior Work with Recurrent Neural Networks in Physiology

RNNs have found use in a wide variety of disciplines and problems, including the healthcare and physiological time series and sequences. RNNs trained on time-stamped medical events, such as medication orders or disease diasnoses, have been employed for disease prediction[2] [14]. More specific to blood pressure prediction, Sideris et al. [19] has been able to produce systolic blood pressure estimations and forecasts using only finger pulse oximeters .

2.2.1 Learning to Diagnose with LSTM

Of particular note and influence to this thesis is Lipton et al.'s work which used a Long Short-term Memory (LSTM) RNN in order to classify multivariate time series into a possible 128 diagnoses [12]. Lipton et al. trained their network on body temperature, heart rate, diastolic and systolic blood pressure, and blood glucose time series which were subject to noise and irregular sampling. These time series were aggregated to 1-hour intervals of mean values and scaled to [0,1] intervals using ranges defined by clinical experts. The intent of this work was to classify these time series into using any combination of 128 possible diagnoses, such as acute respiratory distress, congestive heart failure, or seizures. These time series were input into a LSTM RNN with two recurrent layers of 128 hidden units and trained to minimize the log loss of the data, that is:

$$loss(\hat{\boldsymbol{y}}, \boldsymbol{y}) = \frac{1}{|L|} \sum_{l=1}^{l=|L|} -(y_l \cdot \log(\hat{y}_l) + (1 - y_l) \cdot \log(1 - \hat{y}_t)) \tag{2.1}$$

where $\boldsymbol{y_1}$ is the true binary vector representing the possible 128 diagnoses for example l, $\hat{\boldsymbol{y}}_{\boldsymbol{l}}$ is the model's prediction, and L is the entire set of examples.

In order to produce a prediction, Lipton et al.'s model only needed to output the 128-vector of predicted diagnoses once after processing the entire input. In order to improve the model's performance, however, they leveraged LSTM RNNs' ability to output a prediction at every time step and employed a technique called target replication which instead uses a loss of

$$\alpha \cdot \frac{1}{T} \sum_{t=1}^{T} loss(\hat{\boldsymbol{y}}^{\boldsymbol{t}}, \boldsymbol{y}^t) + (1 - \alpha) \cdot loss(\hat{\boldsymbol{y}}^T, \boldsymbol{y}^T) \tag{2.2}$$

where the contribution of all non-final predictions is given a weight α. The change is depicted in Figure 2.2.1. Target replication forces the model to produce an accurate prediction as early as possible, leading to more efficient propagation of error. Finally, in order to regularize the model, Lipton et al. employed dropout regularization and l_2 weight decay.

We have adopted many of the modeling considerations made in [12] and will delve further into a description of LSTM RNNs, their training, and dropout regularization in Section 3.

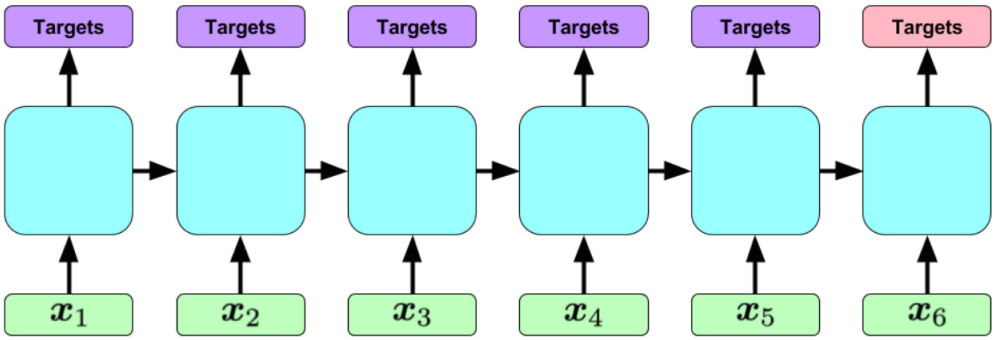

Figure 2-1: An example of a RNN classification model with target replication. All targets are used during training time, but only the final target (in red) is considered for evaluation.

Chapter 3

Experimental Design and Data Preparation

The following chapter details the design of the model, its training procedure, and data preparation steps.

3.1 Model Design and Training Considerations

As explored in the previous chapter, there are a dizzying variety of models and methods with which to tackle the AHE classification problem. This section details the model design and specific optimizations employed to train the model.

3.1.1 Recurrent Neural Network Overview

For these experiments, we employed recurrent neural networks (RNNs). Recurrent neural networks seemed well suited for the task of classifying AHE for a few key reasons. RNNs are well-known to work well for learning tasks where the input data is sequential. Sharing weights between hidden units across each time step, the RNN architecture is a natural way to model time series data, where each time step of the input depends on those in the past. Furthermore, RNNs have the capability to handle variable-length inputs, eliminating the need for padding inputs. In contrast,

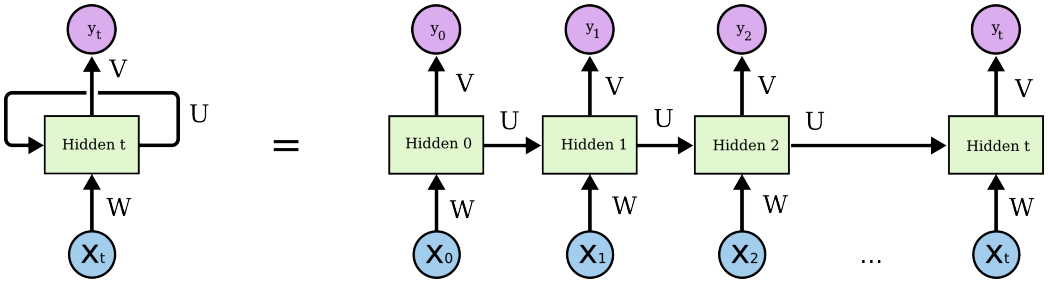

Figure 3-1: Example of Unrolling a RNN over the input time steps. Weight sharing across time steps is encapsulated in the recurrent connection.

multilayer perceptrons or convolutional neural networks are constrained to fixed-size inputs, fixed-size outputs, and a fixed-number of layers and computational steps.

Traditional RNNs consist of an input layer, an output layer, and a recurrent layer, as depicted in 3-1. They are comprised of a series of weight matrices and activation functions. Explicitly, the set of equations that maps a set of inputs, $\boldsymbol{x}$ to predicted outputs $\hat{\boldsymbol{y}}$ is:

$$\boldsymbol{h_1} = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{b_h})$$
$$\boldsymbol{h_i} = \sigma(\boldsymbol{W}\boldsymbol{x} + \boldsymbol{U}\boldsymbol{h_{i-1}} + \boldsymbol{b_h})$$
$$\hat{\boldsymbol{y}}_{\boldsymbol{i}} = softmax(\boldsymbol{V}\boldsymbol{h_i})$$

Where σ is the logistic function,$\boldsymbol{b}$ are bias vectors, and $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}$ are weight matrices shared across time steps. Over the course of training, the model learns which setting of weight matrices $\boldsymbol{W}, \boldsymbol{U}, \boldsymbol{V}$ will minimize an overall loss function. In this work, we used the log loss (Eq. 2.1) and shall assume that to be the loss function for the rest of this thesis.

3.1.2 Optimizing the Weight Matrices

Generally speaking, neural networks employ some form of minibatch stochastic gradient descent (SGD) to learn an appropriately good setting of the weight matrices with respect to some loss function, $\mathcal{L}(x)$. The minibatch SGD update with respect

to a weight parameter θ is:

$$\theta_{t+1} = \theta_t - \lambda \sum_{s \in \boldsymbol{X}} \frac{\delta \mathcal{L}(x)}{\delta \theta_t}$$

where the sum is taken over some subset of the training examples, X and λ is some user-set parameter. Finding the network's gradients is accomplished via the backpropagation through time algorithm, which in essence unrolls the recurrent network in time, as depicted in 3-1, and applies the standard backpropagation algorithm to the resulting network. A full description of the algorithm can be found in Mozer et al. [16].

3.1.3 RMSProp and Stabilizing the Effective Learning Rate

Minibatch SGD by itself is poorly suited for training RNNs. The method is extremely sensitive to the choice of learning rate. Too large, and learning is unstable with parameters oscillating wildly around a local optimum. Set it too small, and learning becomes intolerably slow. Furthermore, a single learning rate that was well suited early in optimization may end up non-ideal somewhere else in the parameter space. There are numerous adjustments to minibatch SGD that seek to remedy these issues. We choose rmsprop.

An rmsprop update consists of the following two steps:

$$\mathbb{E}[g^2]_{t+1} = 0.9\mathbb{E}[g^2]_t + 0.1g_{t+1}^2$$
$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathbb{E}[g^2]_{t+1} + \epsilon}}$$

Where g_t is the minibatch gradient calculated at iteration t of training with respect to parameter θ_t, $\mathbb{E}[g^2]_t$ is an approximation to the running average of squared gradients, and η is user-set hyperparameter that adjusts the rate of learning. In order to keep the parameter update roughly consistent, the algorithm scales the learned gradient by the square root of a running average of squared gradients $\sqrt{\mathbb{E}[g^2]_t}$. The

running average as defined here gives much higher weight to recently calculated gradients which are more likely to have a magnitude similar to the one at the current time step, and thus better ensures that the parameter is updated by a consistent amount. By ensuring the size of the actual parameter update is consistent, learning will be more stable without the need for a cripplingly slow learning rate, and thus more rapid. For the purposes of this work, we set $\epsilon = 10^{-6}$ and $\eta = 10^{-5}$.

3.1.4 Long Short-term Memory Units

Though RNNs share their weights across each time step and may not have a large number of parameters, they are still very deep networks. Error must be propagated over many steps to compute gradients with respect to inputs early in a long sequence. Because of this, should gradients be significantly smaller or larger than 1, final gradients will shrink or explode. In this work, we employ RNNs with sequence lengths of 900, far beyond the capacity of traditional RNNs. We can mitigate this issue by employing long short-term memory (LSTM) units in our recurrent layers which will allow error to propagate throughout the entire network. Long short-term memory units come in many forms, but all of them have some form of input, forget, and output gates [9]. They often will also come with "peephole connections" as presented in [4] which expose the internal state to the gates.

The architecture of the LSTM unit used in this work is depicted in 3-2 and encapsulated in the following equations:

$$
\begin{aligned}
g_l^{(t)} &= \phi(W_l^{gx}\mathbf{h}_{l-1}^{(t)} + W_l^{gh}\mathbf{h}_l^{(t-1)} + \mathbf{b}_l^g) \\
i_l^{(t)} &= \sigma(W_l^{ix}\mathbf{h}_{l-1}^{(t)} + W_l^{ih}\mathbf{h}_l^{(t-1)} + W_l^{ic}c_l^{(t-1)} + \mathbf{b}_l^i) \\
f_l^{(t)} &= \sigma(W_l^{fx}\mathbf{h}_{l-1}^{(t)} + W_l^{fh}\mathbf{h}_l^{(t-1)} + W_l^{fc}c_l^{(t-1)} + \mathbf{b}_l^f) \\
o_l^{(t)} &= \sigma(W_l^{ox}\mathbf{h}_{l-1}^{(t)} + W_l^{oh}\mathbf{h}_l^{(t-1)} + W_l^{oc}c_l^{(t-1)} + \mathbf{b}_l^o) \\
c_l^{(t)} &= g_l^{(t)} \odot i_l^t + c_l^{(t-1)} \odot f_l^{(t)} \\
h_l^{(t)} &= \phi(c_l^{(t)}) \odot o_l^{(t)}
\end{aligned}
$$

Each element of the equations above is summarized as follows:

- W_l^{mn} corresponds to weight matrix feeding into the lth hidden layer of LSTM units, starting from the n gate and going to the m gate.
- g is a a transformation of incoming data from the previous layer
- i is an input gate. Setting it closer to 1 makes the unit more likely to accept the new input
- f is the forget gate. Setting it closer to 1 makes old values of s matter less.
- c is the error carousel, which maintains state from time step to time step
- o is the output gate. Setting it closer to 1 makes the LSTM unit more likely to broadcast its activation.
- x in the superscript of the weight matrix W denotes the input vector for this LSTM unit. It can come either directly from the input at the current time step, or a previous layer.
- b denotes the bias vector.
- $\odot$ is the hadamard product, and superscripts in regard to t denote the time step associated with the gate or unit.

LSTM's ability to learn long-term dependencies resides in its ability to learn how to access and alter the error carousel. Whereas a traditional RNN's setting of $\boldsymbol{h_t}$ is liable to face noise which interferes with error propagation, LSTMs can learn which inputs are meaningful given the additional context of what's already in the error carousel and previous hidden states. Through the use of input, forget, and output gates, it can learn when incoming information is important, when it should be integrated into the internal state of the cell, and when it should propagate its internal state.

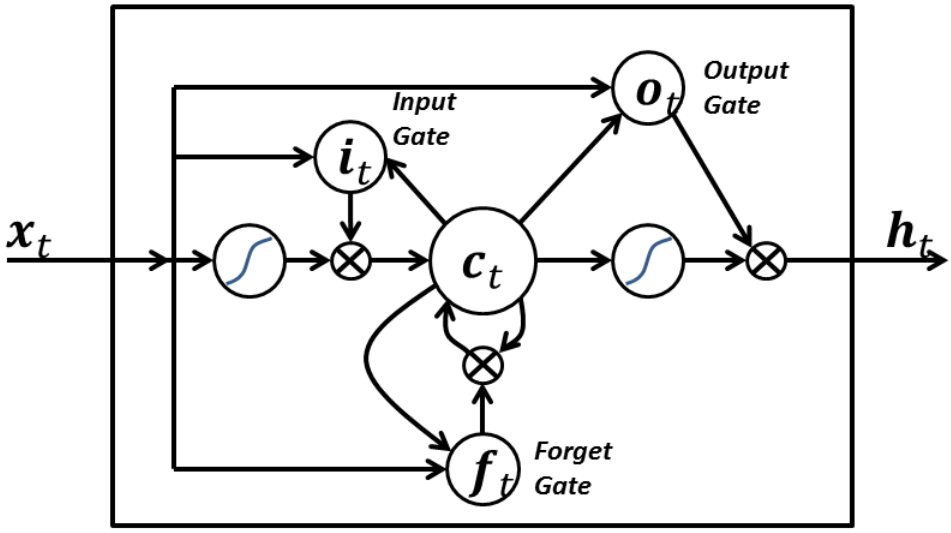

Figure 3-2: A depiction of the LSTM unit used in this work. Connections between hidden units across time steps have been omitted for clarity. Figure adapted from [22].

3.1.5 Dropout Regularization

Large neural network models contain many parameters and have the capacity to model extremely complex functions. This capability is a blessing and a curse. Such models will often overfit on the training set and lose generalizability and accuracy on the test set. We employ dropout regularization with a dropout probability of $p = 0.5$ in the output layer and between recurrent layers to combat this. Dropout regularization independently sets each weight in consideration to zero with probability p. In response to this, the network cannot rely on a few weights per-example to predict an outcome, lest those weights get pruned in a training step. The model is thus forced to employ many weights to process and predict each example, reducing overfitting. We deliberately avoid dropout on weights between time steps, as doing so effectively eliminates long-range memory.

3.1.6 Gradient Clipping

Though LSTMs are designed to combat the issues of exploding and vanishing gradients, they are still susceptible to this persistent problem[1]. For this reason, we

ensure that the gradient never exceeds a magnitude of 100 for any single parameter to add additional stability to training.

3.1.7 Final Model Architecture and Optimization Procedure

With all these considerations in mind, we build a network as depicted in 3-3. A univariate time series with value x_i at time step i feeds into a LSTM layer, with hidden units ranging in number from 64 to 512. Depending on the model architecture, this layer may feed into a second recurrent layer of the same size, or pass directly to the output layer. The final time step of the final recurrent layer feeds into a softmax output layer, which outputs the probability that the input corresponds to an AHE episode. Error is evaluated as the log loss of the output data and parameters are updated via the rmsprop algorithm with 20 examples per minibatch.

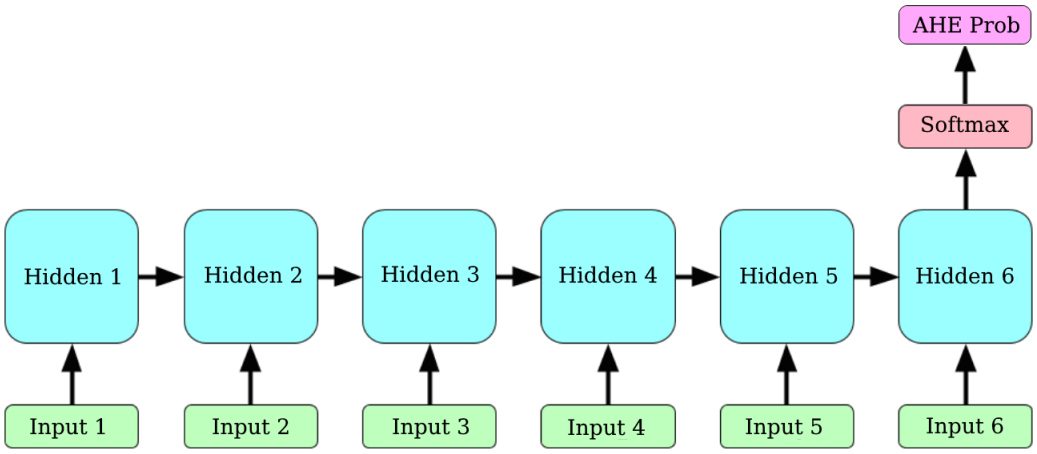

Figure 3-3: Final Model Architecture for the case of a single hidden layer and a six-interval long input time series. Note that the hidden blocks contain anywhere from 64 to 512 LSTM units.

3.2 Data Preparation

We explore our different network configurations using an ABP dataset derived from the MIMIC II dataset. The dataset was pre-prepared by the ALFA group at CSAIL.

Each raw patient waveform is passed through a beat onset detection algorithm and validity detector. Features like mean, duration, and validity for each beat are calculated and coalesced into time-aligned sequences for each patient.

At this juncture the dataset is comprised of about 6000 different groupings of beat-by-beat feature time series. Each grouping contains three corresponding files, called mean.txt, duration.txt, and validation.txt. Mean.txt contains the beat-by-beat means, calculated as the area under the blood pressure curve. Duration.txt contains the duration of each beat, measured in number of 125 Hz samples. The raw time series contain faulty measurements on account of the monitoring equipment, so it is also important to only consider data points we are confident are valid. The third file, validation.txt contains flags on each beat, labeling them as valid or not. Explicitly, if a patient has the values 65, 100, and 1 at the hundredth position in each of the three files associated with him, then the hundredth beat measured from him has a mean pressure of 65 mmHg, duration of 100 samples or 0.8 seconds, and is a valid measurement.

We then split the files into a set of error-free time series by only including time series that had at least 7000 beats with an uninterrupted run of 1's in the corresponding validation.txt file. We choose the length of 7000 as a cutoff as an early filtering step. AHE-positive time series with less than this amount generally are filtered out downstream on account of not having enough beats once the intervals in the prediction window and lead were removed. Note that by doing this split, a single patient may have had more than one time series present in the final training dataset if there was a single faulty pressure reading between long runs of otherwise valid beats. Next, we label the time series for AHE using a definition slightly adjusted from the PhysioNet challenge. Namely, our version of an AHE corresponds to any span of beats in a 30-minute time interval where over 90% of the means are below some threshold value. For our purposes, the threshold is set between 58 and 64 mmHg. If we find a beat segment corresponding to an AHE in a time series, we label the time series as AHE-positive then remove the AHE and all beats occuring within a 30-minute time window preceding the event. If no AHE is detected in a time series, it is left unaltered.

This leads to a slight imbalance in series lengths between the two populations, with AHE-negative examples having on average about 600 more beats. This does not bias the data, however, since LSTM RNNs are agnostic to input length.

Finally, we need to compress these time series to an appropriate length for our RNN. Although LSTM RNNs in theory can handle arbitrarily long sequences of input data and are explicitly designed to combat the exploding and vanishing gradient issues, the model still has difficulty learning very far-ranging interactions [6]. To ensure our networks can propagate information across all time steps in a rapid fashion, we shorten the time series by aggregating consecutive beat-by-beat mean measurements into intervals of either approximately six seconds or six beats.

At this point, the data set is in its final form: a univariate time-series of six-second or six-beat mean blood pressure measurements. We feed these measurements along with their predictions (0 for non-AHE, 1 for AHE) into the previously-described RNN architecture and extract various performance metrics using five-fold cross-validation. Our dataset is heavily biased towards negative examples, and our model initially tends to reject almost all inputs. In order to bias the model to a higher sensitivity, we duplicate AHE-positive examples in the training dataset so that there is an equal number of positive and negative examples. We find empirically that our particular models all converge after 25 training epochs, and thus train all of our models for that duration. Depending on the experiment, each model is trained on 4800 examples, of which 200-320 are unique AHE-positive, and evaluated against around 1250 examples, of which only about 60 are AHE-positive.

Code associated with generating the data pipeline above and training the LSTM RNN model can be found and experimented with at https://github.mit.edu/ALFA-Gigabeats/NN_MEng_Alex_Jaffe.

Chapter 4

Results and Conclusions

This section covers our model's performance as we adjust the number of hidden LSTM units in our model, the lead time, the threshold with which we determine an AHE, and type of time-series aggregation.

4.1 Metric Definitions and Clarifications

In the following section we make extensive use of the terms recall, precision, and sensitivity. Recall is defined as

$$R = \frac{TP}{TP + FN} \tag{4.1}$$

Where TP is the number of true positive examples (that is, examples that are AHE-positive and are labeled by the model to be AHE-positive) and FN is the number of false negatives. Recall gives us a sense for how good our model is at detecting AHE-positive examples. Precision is

$$P = \frac{TP}{TP + FP} \tag{4.2}$$

and gives us a sense for how likely a positive label from the model is accurate. Finally, sensitivity is defined as

$$S = \frac{TN}{TN + FP} \tag{4.3}$$

and provides a feel for how accurate the model is at ensuring negative examples are considered negative.

4.2 Performance of Different Layer Sizes

We first wanted to get a general sense for how well the model performed as we adjusted the size and number of recurrent layers in the network. Models were trained for 25 epochs on a training set of about 4800 examples, 200-300 of which were unique AHE examples. The training set consisted of 1250 examples and had an AHE-positive to AHE-negative ratio of 1:20. Unless otherwise noted, statistics were obtained from five training runs of the model using five-fold cross-validation. We detail the performance of our models with regard to recall, specificity, and precision in the table below.

Model Size	Stat	Recall	Specificity	Precision
1x512	mean	0.76	0.978	0.64
	median	0.76	0.972	0.57
	min	0.74	0.967	0.55
	max	0.79	0.986	0.72
2x64	mean	0.78	0.973	0.59
	median	0.77	0.98	0.65
	min	0.72	0.948	0.44
	max	0.87	0.982	0.69
2x128	mean	0.77	0.977	0.63
	median	0.76	0.979	0.65
	min	0.72	0.967	0.55
	max	0.82	0.984	0.72
2x256	mean	0.77	0.974	0.60
	median	0.75	0.977	0.62
	min	0.74	0.965	0.56
	max	0.82	0.983	0.69

Table 4.1: Summary of performance across model sizes when trained on time series data with a 30 minute lead time.

The models across different sizes possessed largely similar performance in terms of recall, specificity and precision. Though specificity hovers at 97.5% and recall at

78%, due to the population imbalance of 1:20 non-AHE to AHE episodes, precision suffers. Still, comparing against the thresholding technique in Mneimneh et al. [15], we find that their model would yield a precision of about 33% given our population distribution.

4.3 Performance Over Different Lead Times

Next we investigated the performance of the 2x128 model with respect to varying lead times. We chose the 2x128 model from the previous section as it performed nearly as well as the 1x512 model with the use of half as many hidden units. It is important to note the training set sizes differed in this section. Smaller lead times allowed more AHE examples to be present, with the number of unique examples in the training set doubling to about 500. Likewise, increasing the lead time to 45 caused us to lose some AHE examples, the total number dropping to about 150 from 250. Results over leads spanning from 0 to 45 minutes are tabulated below.

Lead Time	Stat	Recall	Specificity	Precision
0	mean	0.95	0.985	0.83
	median	0.95	0.987	0.79
	min	0.93	0.981	0.73
	max	0.96	0.992	0.87
15	mean	0.8	0.973	0.44
	median	0.84	0.95	0.47
	min	0.70	0.935	0.39
	max	0.85	0.968	0.58
30	mean	0.77	0.977	0.63
	median	0.76	0.979	0.65
	min	0.72	0.967	0.55
	max	0.82	0.984	0.72
45	mean	0.78	0.954	0.49
	median	0.81	0.956	0.49
	min	0.69	0.944	0.42
	max	0.88	0.982	0.68

Table 4.2: Summary of performance with different lead times before AHE. All models are trained on 2x128 hidden unit networks with a lead time of 30 minutes.

From the lead time of 0, we can get a baseline for just how well the model can

perform in its current state. It seems that the model has difficulty obtaining higher than 83% precision consistently. Though the model may simply be unable to predict any better than this apparent bound, this non-perfect performance may also arise from a peculiarity in the way AHE is defined; AHE is defined over an interval in which 90% of beats are below some threshold. Should a patient's blood pressure take a sudden drop, then the drop in blood pressure may fall within the beginning of the AHE window and be hidden from the model. A second interesting trend is the relatively poor performance of the model at the 15-minute mark. This may have arisen due to the increased number of examples present to the model causing it to require more than the allotted number of epochs to converge.

4.4 Performance Over Different AHE Thresholds

We also investigated how model performance varies as we alter the blood pressure threshold with which we define an AHE. Since hypotension arises for patients at different blood pressures, we need to ensure that the model is somewhat robust against changes of the definition of AHE. Thus, we also tested to see how the models performed if we instead defined AHE to occur at blood pressure levels of 58 mmHg and 64 mmHg. The results of these trials on 2x128 and 2x256 hidden unit models are detailed in Table 4.3.

Threshold (mmHg)	Model Size	Recall	Specificity	Precision
58	2x128	**0.82**	**0.992**	**0.83**
58	2x256	0.79	0.98	0.66
60	2x128	0.77	0.977	0.63
60	2x256	0.77	0.974	0.60
64	2x128	0.71	0.975	0.59
64	2x256	0.71	0.99	0.78

Table 4.3: Mean recall, specificity, and precision over different thresholds. Lead time is fixed at 30 minutes.

Two trends arise in the table above. First, by making our definition of AHE more stringent, the model is better able to discriminate an AHE. Second, is that by raising the AHE threshold, recall diminishes. This suggests patients who exhibit

more extreme cases of AHE in regard to blood pressure also exhibit more discernible patterns in the hour leading up to the event.

4.5 Performance with Beat Compression

As a last trial, we tested to see if compressing the time series to six-beat intervals would alter the performance of our models. A comparison table is below.

Beat Compressed	Model Size	Recall	Specificity	Precision
Y	1x512	0.78	0.97	0.58
Y	2x128	0.77	0.97	0.56
Y	2x256	0.78	0.97	0.57
N	1x512	0.76	**0.978**	**0.64**
N	2x128	0.77	0.977	0.63
N	2x256	0.77	0.974	0.60

Table 4.4: Comparison of mean recall, specificity, and precision across different beat compression schemes. Lead time is fixed at 30 minutes.

The primary difference is a reduction in the model's precision. Though we are unsure of the exact mechanism through which this could occur, we do know that aggregating by beats instead of time will effectively stretch out periods of rapid heart rate. This rapid heart rate often coincides with lower blood pressure and thus method of compression could lead to greater distortion of AHE-positive examples than negative. This distortion may obfuscate a pattern in the time series discriminating AHE positive and negative examples.

Chapter 5

Future Directions

There has already been a wide variety of approaches taken for the task of classifying AHE and there are plenty of approaches yet to take stemming from the use of RNNs alone.

5.1 Further Parameter Exploration

Due to hardware and time limitations, we were unable to explore a more complete set of parameters over this problem. We suspect with further parameter tuning our model's recall and precision may improve even more. There are quite a few ways one could tune the training procedure.

5.1.1 Adjust the Proportion of AHE Examples

During training it was observed that the models' recall was inversely related to precision, though improvements to recall came at a very slight deficit in precision. Increasing the proportion of AHE examples further would cause it to value recall higher in its training and in turn lead to higher recall. On the other hand, it may be the case that our proportion of positive examples was too high, and the model could obtain comparable recall and better precision with a less biased dataset.

5.1.2 Experiment with Optimization Procedures

RMSProp was chosen on account of its ability to adaptively adjust the effective rate of learning in response to the size of gradients around it, but there still is a fixed learning parameter, η, associated with it. Adjusting this parameter could lead to a modest improvement in optimization performance. In addition to changing the learning parameter, one could also try other, learning-rate agnostic optimization schemes, like adam, and add some form of momentum so training converges faster [11].

5.1.3 Consider Other Beat Features

The input time series for these experiments consisted of univariate beat-by-beat means aggregated into six-second intervals. Although past work has found that measuring MAP or beat-means alone is often sufficient for predicting an AHE, there may be complimentary information contained in a time series of some other measure of the beat. With knowledge of a beat's duration, maximum pressure, or variance, the model may have access to patterns absent in the mean data alone.

5.2 Employ Bidirectional RNNs

The RNNs used here are relatively straightforward in the grand scheme of RNN architectures. Employing a different kind of RNN could capture certain structure that a traditional RNN has difficulty with. For instance, this work and previous studies on AHE prediction before this one have found that measurements closer to the episode are generally more informative then those with a larger lead time. Bidirectional RNNs, which are RNNs that run both forward and backward in time, could exploit this fact. The model can propagate information concerning the end of the time series to hidden units representing the beginning and in turn may be able to provide additional contextual information to patterns detected early in the time series and better predict AHE.

5.3 Employ Convolutional RNNs

Another potential direction is to employ convolutional RNNs. Convolutional Neural Networks (CNNs) are well suited for detecting spatio-temporally local patterns. These networks are similar to feedforward neural networks, but consist of a pooling layer that coalesces spatio-temporally near inputs using some learned function. As observed earlier, different "pooling" approaches in the form of either aggregating our time series into six-beat or six-second intervals led to significantly different results. By feeding the raw beat-by-beat means into a convolutional layer, then passing the output into an LSTM RNN as used in this paper, it may be possible to simultaneously learn local and long-range patterns predictive of AHE.

5.4 Producing AHE with GANs

Looking beyond simple classification, one could also build a generative model that could learn to produce and forecast AHE-like time series. In fact, RNNs are already well-suited for the task of generating time-series data. They have been shown to generate in a step-by-step manner simulated Wikipedia articles, classical music and a multitude of other sequential data [6]. Recently, a new paradigm in neural network training called generative adversarial networks (GANs) has yielded impressive results generating realistic images from other images and text [17] [18]. These networks are split into a generative model that models the data distribution and a discriminative model which estimates the probability that an input game from the generating model instead of the original dataset [5]. Integrating an LSTM RNN with a GAN could enable the production and prediction of raw time series indistinguishable from that

Bibliography

[1] Martin Arjovsky, Amar Shah, and Yoshua Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

[2] Edward Choi, Andy Schuetz, Walter F Stewart, and Jimeng Sun. Using recurrent neural network models for early detection of heart failure onset. *Journal of the American Medical Informatics Association*, page ocw112, 2016.

[3] Franck Dernoncourt, Kalyan Veeramachaneni, and Una-May OReilly. Gaussian process-based feature selection for wavelet parameters: Predicting acute hypotensive episodes from physiological signals. In *2015 IEEE 28th International Symposium on Computer-Based Medical Systems*, pages 145–150. IEEE, 2015.

[4] Felix A Gers, Nicol N Schraudolph, and Jürgen Schmidhuber. Learning precise timing with lstm recurrent networks. *Journal of machine learning research*, 3(Aug):115–143, 2002.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

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

[7] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[8] JH Henriques and TR Rocha. Prediction of acute hypotensive episodes using neural network multi-models. In *Computers in Cardiology, 2009*, pages 549–552. IEEE, 2009.

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

[10] Yongwook Bryce Kim, Erik Hemberg, and Una-May O'Reilly. Stratified locality-sensitive hashing for accelerated physiological time series retrieval. In *Engineering in Medicine and Biology Society (EMBC), 2016 IEEE 38th Annual International Conference of the*, pages 2479–2483. IEEE, 2016.

[11] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] Zachary Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with lstm recurrent neural networks. *International Conference on Learning Representations*, 2016.

[13] Junhua Mao, Wei Xu, Yi Yang, Jiang Wang, Zhiheng Huang, and Alan Yuille. Deep captioning with multimodal recurrent neural networks (m-rnn). *arXiv preprint arXiv:1412.6632*, 2014.

[14] Adam McCarthy and Christopher KI Williams. Predicting patient state-of-health using sliding window and recurrent classifiers. *arXiv preprint arXiv:1612.00662*, 2016.

[15] MA Mneimneh and RJ Povinelli. A rule-based approach for the prediction of acute hypotensive episodes. In *2009 36th Annual Computers in Cardiology Conference (CinC)*, pages 557–560. IEEE, 2009.

[16] Michael C Mozer. A focused back-propagation algorithm for temporal pattern recognition. *Complex systems*, 3(4):349–381, 1989.

[17] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[18] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 3, 2016.

[19] Costas Sideris, Haik Kalantarian, Ebrahim Nemati, and Majid Sarrafzadeh. Building continuous arterial blood pressure prediction models using recurrent networks. In *Smart Computing (SMARTCOMP), 2016 IEEE International Conference on*, pages 1–5. IEEE, 2016.

[20] Richard Socher, Cliff C Lin, Chris Manning, and Andrew Y Ng. Parsing natural scenes and natural language with recursive neural networks. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 129–136, 2011.

[21] Alexander Waldin. *Learning blood pressure behavior from large blood pressure waveform repositories and building predictive models.* PhD thesis, MasterâĂŹs thesis, MIT, 2013.

[22] Wikipedia. Long short-term memory — wikipedia, the free encyclopedia, 2017. [Online; accessed 26-May-2017].