CMPE 258 Final Report

Team Members: Sonia Mannan 009009117, Zoe Lie 010646467, Ryan Choy 014499316

Background

Several concepts such as supervised classification and RNNs are necessary to understand this report. This section will summarize them briefly to give the reader more context.

Supervised Machine Learning

Machine learning is a field in which machines are used to make decisions based on some information without requiring explicit instructions. An example of this is a lending company looking at someone's credit score, age, and savings to determine if they should get a loan. Another example is a real-estate agent using the size of a house, number of bedrooms, and commute time to determine what someone will pay for a house. In these cases the machine can automate the process or inform the company's decision to save time and avoid manually inspecting large amounts of data.

In these examples, known data is gathered beforehand to determine what the decision on unseen data will be. An algorithm can learn a mapping of inputs to outputs by taking the known data and minimizing the loss between estimated and actual values to accurately make decisions on unknown data.

Sequence Classification

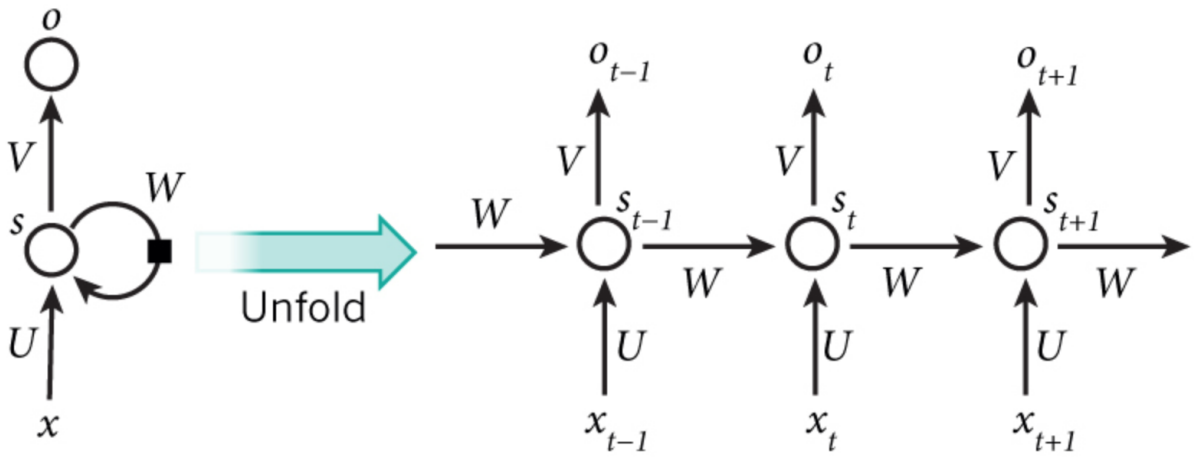
Unlike the lending and home price examples above, some data is sequentially related, for example, the humidity, precipitation, and UV index over the last 10 days, or a movie review in which words in a sentence form a sequence. In this type of data, the order of inputs is important and often related. For example, the ordering of words in a review determines its meaning. Supervised classification tasks can also be performed on sequences of inputs. For example, given the last 10 words in this sentence, is the tone of the paragraph positive, neutral, or negative? Given the last 10 days of humidity, precipitation, and UV readings, will today be sunny or rainy?

Similar to any other supervised classification task, known data is gathered beforehand so the algorithm can learn an optimal mapping of inputs and outputs. For example, we may gather several reviews and manually label their tone or look at historical weather data where we knew when it was sunny or rainy. Some models may extract features from a sequence of inputs, or use the sequence as-is to classify unseen sequences of data.

Recurrent Neural Networks (RNNs)

RNNs are different from other types of neural networks in that they account for the dependencies between inputs and are particularly useful in working with sequential data. Most neural networks assume inputs are independent of each other, which doesn't work for sequential data such as a sentence, where previous words can affect future words in the sentence.

The "recurrent" portion of an RNN refers to the process of applying the same operations on multiple inputs over time, allowing the RNN to retain historical information in the sequence and account for time dependencies.



A recurrent neural network and the unfolding in time of the computation involved in its forward computation. Source: Nature

Figure 1: RNN Diagram [7]

Referring to the diagram shown in figure 1, a new input, x, is given to the model at each timestep, t. The model maintains a hidden state, s, which it uses to update the weights, W, U, and V, and output a new prediction, o, at each timestep. This diagram shows the RNN "unfolded" so the reader can view the operations at each timestep. The RNN uses the same weights across different inputs and timesteps to track historical dependencies in the sequence. A non-linear activation is applied on the output and optionally a bias term, B, is included. Refer to the equations below for how the hidden state and output are calculated at each timestep.

$$S = max(Xt \bullet U + S \bullet W + Bh, 0)$$
$$Ot = max(S \bullet V + By, 0)$$

In a standard neural network, back propagation is used to update weights in order to minimize the network's loss. This is done by minimizing the network's cost function and determining how much to tweak the network's weights. Standard networks optimize this process by using stochastic gradient descent, where the "amount" to tweak the weights is known as the gradients.

RNNs use a modified version of back propagation known as backpropagation through time (BPTT) where the error for each time step is calculated and accumulated backwards in time to update the weights. In this way, the reader can view each timestep as a different "layer" in the RNN.

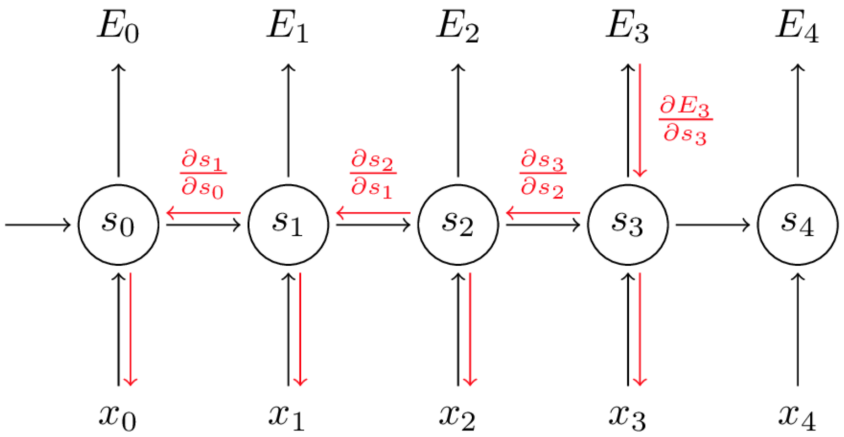


Figure 2: BPTT Diagram [8]

RNNs for Sequence Classification

RNNs retain state across time to track sequential dependencies and are used commonly to analyze time series data. RNNs can also be used in supervised learning tasks to classify sequences of inputs. For example, let's say we want to classify tomorrow's weather as either rainy or not rainy depending on the last 10 days of precipitation, humidity, and UV index where we know how the weather was over the last 10 days. The input at each time step would be 3 variables (precipitation, humidity, and UV index), and the output would be a predicted class (rainy or not rainy). If we have thousands of sequences of 10-day data, we can feed each sequence individually into the network and use BPTT to update our weights in order to minimize the classification error. This way, we develop a network which can accurately predict if it is going to rain tomorrow or not.

Introduction

In this project, our main goal is to explore and compare the performances between various variations of the Recurrent neural network at predicting human activity from smartphone sensor data. These models include regular LSTMs, Convolutional LSTMs, Stacked LSTMs, Bidirectional LSTMs, and GRUs. We want to see if there was any kind of advantage one model had against other ones, whether it's worth the hassle of implementing a more complex model rather than a simpler one that produces similar results. We want to analyze why some models

perform better than others. We will be testing out the recurrent neural network models on the UCI HAR dataset which will be further explained in the upcoming sections.

Related Work

We used the University of California Irvine's (UCI) Human Activity Recognition (HAR) dataset, which was collected in 2012 as part of a research project to better understand human activities from smartphone sensor signals [9]. Since then, a variety of research on this dataset has been conducted to improve classification performance of activities from acceleration and velocity signals.

The original paper published from this dataset in 2013 by Chetty et. al manually curated a list of 561 features from the sensor readings and compared the performance of several supervised classifiers such as SVMs, Naive Bayes, and Random Forests, on the features. They also experimented with an IBk lazy-learner. IBk does not build a model and can avoid expensive training times, but makes predictions by comparing points to their nearest neighbors, which can be slow.

The engineered features included metrics such as the min, max, mean, or correlation between two signals over time, and more. Chetty et al evaluated model performance using a 5-fold cross validation on the 10,000 samples collected, inputting the 561-length feature vector and class label to the model to get a prediction. They collected their results in the following table, testing each model on an increasing number of features and recording the time to train. The team found that in terms of efficiency and accuracy, the random forest model performed better than a single decision tree, Naive Bayes, or K-Means, achieving over 96% accuracy with the full feature set. The team also found that an IBk lazy-learner had the best prediction accuracy. Even though the "training" time for this algorithm is low, the prediction time tends to be high, so the random forest model was preferred.

Num Feat	KM	NB	J48	RF	RC	IBK
2	38.00	49.45	56.30	55.60	60.10	53.18
8	68.40	48.26	61.39	63.01	63.03	60.18
16	69.00	48.57	69.02	71.27	71.10	67.84
32	70.00	52.34	70.24	74.17	75.10	71.74
64	59.00	56.10	77.30	77.51	83.73	77.51
128	59.50	55.31	91.46	94.29	95.10	92.97
256	57.00	53.86	93.81	95.63	96.28	97.55
561	60.00	79.00	94.00	96.30	96.90	97.89

(a)

Num Feat	KM	NB	J48	RF	RC	IBK
2	15.1	0.0	0.9	7.3	14.4	0.0
8	20.6	0.0	7.4	16.8	17.7	0.0
16	37.4	0.3	11.4	19.7	23.4	0.0
32	67.9	0.9	25.7	25.7	25.4	0.0
64	119.4	1.7	38.0	29.1	31.5	0.0
128	217.0	4.4	64.6	31.7	30.7	0.0
256	457.5	3.3	52.7	20.1	25.7	0.1
561	582.1	5.8	247.4	14.7	27.0	0.5

(b)

Fig. 2.Comparison of Classifier Performance: (a) Recognition Accuracy; (b) Model Building Time

Figure 3: Chetty et al supervised classification results

The results in Chetty et al's original work were promising but require a lot of manual feature engineering on the signal data. Since 2013 and the rise of wearable devices, many other

researchers have tried to improve on this through the use of neural networks. Ullah et al developed a stacked LSTM approach in 2019. Their network consisted of a ReLu layer to normalize the sensor data, followed by five LSTM layers, and a final softmax layer to output predictions. Ullah et al also used a standard cross entropy loss function with an extra L_2 regularization term to add bias into the model and prevent overfitting and an adam optimizer to update weights. They used the UCI HAR dataset, splitting the data into a 70:30 train split where each sample was a 128x9 vector containing 128 time steps and 9 measurements taken per time step.

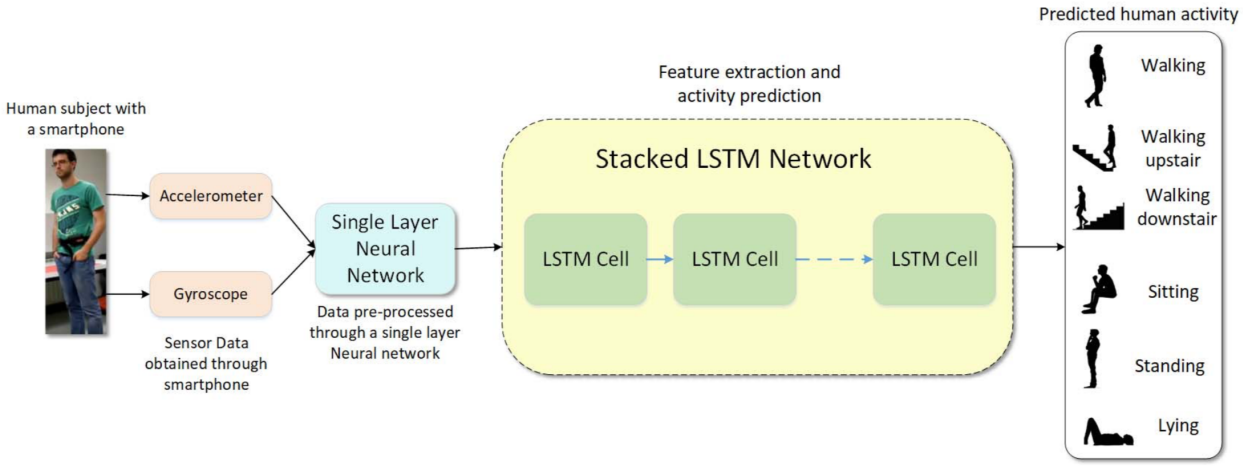


Figure 4: Stacked LSTM Architecture, Ullah et al

Ullah et al compared their results, achieving over 93% accuracy, to other Markov and SVM models; however they were not able to reproduce the original 96% accuracy presented by Chetty et al in 2013, although they were able to produce results with significantly less feature engineering.

Table 1: Ullah et al Average Accuracy Results

Techniques	**Average Accuracy**
Kim et al. [22]	83.51
Anguita et al. [24]	89.00
Ronaoo et al. [23]	91.76
Ronaoo et al. [35]	90.89
Seto et al. [25]	89.00
Li et al. [28]	92.16
Proposed	93.13

When comparing individual class performance of their model, Ullah et al found that the model performed best at predicting when a participant was walking and struggled to differentiate between sitting and standing, but had an overall high precision and recall across all activities.

Table 2: Ullah et al Stacked LSTM Confusion Matrix

		Predicted behavior						
		walking	walking upstairs	walking downstairs	sitting	standing	laying	Recall
Actual behavior	walking	467	3	26	0	0	0	94.15%
	walking upstairs	7	438	26	0	0	0	92.99%
	walking downstairs	10	3	407	0	0	0	96.90%
	sitting	0	24	2	596	29	0	91.50%
	standing	0	3	0	47	402	0	88.93%
	laying	0	27	0	0	0	510	94.97%
	Precision	96.48%	87.95%	88.28%	92.69%	93.27%	100%	

In addition to stacked LSTMs, other researchers have also proposed using bidirectional LSTMs to further improve results. Bidirectional LSTMs analyze the input sequence going forward and backwards in time, leveraging future inputs to better inform a decision at the current timestep. This approach was proposed by both Hernandez et al in 2019 at the STSIVA Symposium and by Yu et al at the 2018 International Conference on Mechanical, Control and Computer Engineering.

Similar to Ullah et al, Hernandez et al also used an Adam Optimizer to update weights and a grid search approach to search for the optimal number of LSTM layers and neurons per LSTM. They used a learning rate of 0.001 and inputted a feature vector of 128x9 (timesteps x features) to the model. They found that a deeper network with 3 layers and 175 neurons per LSTM performed best, at 92.67% average accuracy after 500 epochs of training.

Table 3: Hernandez et al BiLSTM Average Accuracy Results

TABLE III
ACCURACY ON THE TEST SET FOR DIFFERENT L AND N. ACCURACY IS EXPRESSED AS A PERCENTAGE. THE HIGHEST ACCURACY IS HIGHLIGHTED IN BOLD.

L	N						
	100	125	150	175	200	250	300
1	92.06	90.50	91.35	90.77	90.26	89.21	90.40
2	89.41	90.46	90.50	91.55	90.70	90.87	89.72
3	69.70	88.06	90.80	**92.67**	89.45	87.21	88.12

Similar to Ullah et al, Hernandez et al also found that the model had the most difficulty classifying standing from sitting. Although Hernandez et al's average accuracy of 92.67% was close to Ullah's 93%, it seems that bidirectional layers performed slightly worse than regular, stacked LSTMs.

Table 4: Hernandez et al Stacked BiLSTMs class precision and recall

TABLE V
CONFUSION MATRIX ON THE TEST SET FOR THE BEST CLASSIFICATION PERFORMANCE.

	W	WU	WD	ST	SD	LD	**REC** (%)
W	488	3	5	0	0	0	**98.4**
WU	1	448	22	0	0	0	**95.1**
WD	0	0	420	0	0	0	**100**
ST	0	25	0	375	91	0	**76.4**
SD	5	2	0	62	463	0	**87.0**
LD	0	0	0	0	0	537	**100**
PRE (%)	**98.8**	**93.7**	**94.0**	**85.8**	**83.6**	**100**	**92.67**

Data

We used the University of California Irvine's (UCI) Human Activity Recognition (HAR) dataset [9]. In this experiment researchers asked 30 participants aged 19-48 to perform six activities: WALKING, WALKING_UPSTAIRS, WALKING_DOWNSTAIRS, SITTING, STANDING, LAYING, while wearing a Samsung Galaxy SII on their waist. While performing the activities sensor data from the accelerometers and gyroscopes from the smartphone were collected. The sensor data was collected at 2.56 second intervals with a 50% overlap, for a total 128 readings per activity. The published dataset is already pre-processed by applying noise filters to the accelerometer and gyroscope sensor data. Participants were asked to perform the activities multiple times so that researchers could collect over 10,000 samples evenly distributed over each activity. Figure 5 visualizes the data by graphing an example signal of the standing activity, and figure 6 shows a graph of the distribution of each activity in the dataset to show that the amount of data for each activity is approximately equal.

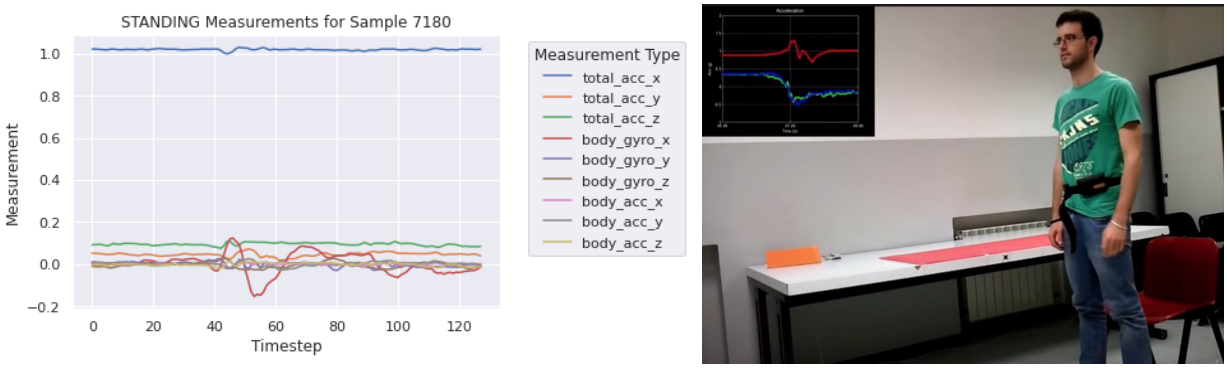


Figure 5: Signals and Image of a Participant Performing the STANDING Activity

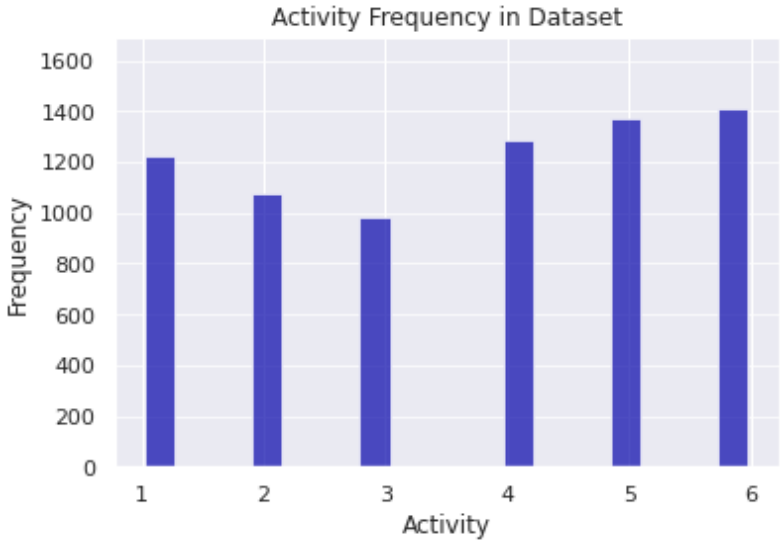


Figure 6: Activity Distribution of the Dataset

Each sample consists of three measurements: total acceleration, body acceleration, and gyroscope velocity, where body acceleration was estimated by subtracting gravitational acceleration from the total acceleration. Additionally, each measurement was taken in respect to three dimensions: X, Y, and Z. The combination of measurement and dimension results in 9 features collected for each time step, where 128 time steps were recorded for each activity.

There were 10,299 samples total and a 70:30 train test split was used resulting in 7,352 training samples and 2,947 testing samples. Each sample is an independent time series corresponding to an activity and containing 128 time steps. Refer to the equations below for the training and testing dimensions:

$$Training\ dimensions: 7352 \times 128 \times 9$$
$$Testing\ dimensions: 2947 \times 128 \times 9$$

The training data was inputted into various LSTM and GRU models in batches of 64 samples each to perform supervised classification on the sequential data where each input was a 128x9

feature vector. For example a forward pass for one sample through the network with a batch size of one would look like this:

1. For each sample in the batch
    a. For each timestep, t: This will be repeated 128 times because there are 128 timesteps per activity
    b. Gather readings at t, X_t: This will be a vector of length 9 because there are 9 sensor readings per timestep
        i. Make a prediction about the activity based on the current readings: Refer to the section "Recurrent Neural Networks (RNNs)" for equations. Equations differ for an LSTM or GRU, but the RNN section provides a base understanding.
    c. After all the time steps are processed, perform BPTT to update the RNN's weights based on the predicted output after the last timestep. In our case, we only care about the prediction from the final time step to tell us what activity the time series corresponds to.

This process is performed thousands of times for all training samples until the weights in our RNNs are optimized to minimize the classification error between true and predicted activities. After this process, the model should be able to accurately determine which of the six activities that the participant is performing based on a time series of signal readings from a smartphone.

Methodology

In this project, we will be exploring multiple variations of recurrent neural networks to train and test the data that was mentioned. The models that we have chosen for this project are: basic LSTM, Convolutional LSTM, stacked LSTM, bidirectional LSTM, and GRU. Our models use a many to one prediction on the dataset. There are 9 inputs (x, y, z coordinates of the total acceleration, body acceleration, and gyro velocity), and the output is the classification of one of the six activities (walking, walking upstairs, walking downstairs, sitting, standing, laying down).

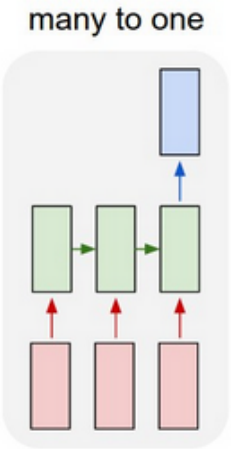
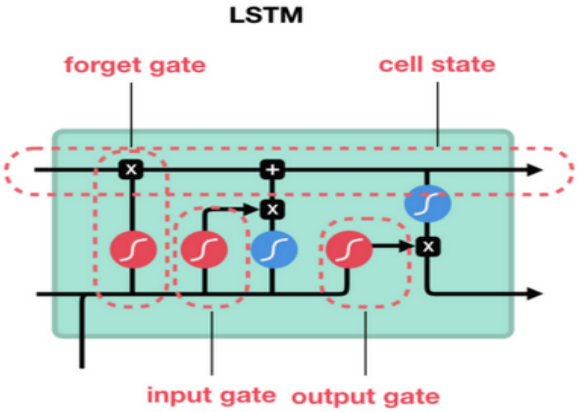


Figure 7: Many-to-one Model

LSTM (Long short-term memory)

Vanilla RNNs suffer from short-term memory, which means that when a sequence becomes too long, it will be harder for the model to carry information from earlier timesteps. Vanilla RNNs will leave out information in the beginning, which is not ideal. The solution to this problem is the introduction of LSTMs. LSTMs have an internal hidden mechanism called gates that regulate the flow of information inputted into the model. These gates are able to learn which data in a sequence is important enough to keep and what to leave out. The cell state and its various gates (forget gate, input gate, output gate) are core concepts for the LSTM. Relevant information is transported through the cell state throughout the processing of the sequence. As information gets passed through, the neural network decides which information to keep and which to leave out. The risk of having vanishing gradients is also decreased. Basic LSTMs have a single hidden layer of LSTM units. They can support multiple parallel sequences of input data, such as the accelerometer and gyroscope data in our dataset.

Below is an illustration of the LSTM model and the descriptions of its gates:



- Forget Gates
    - Decides what information to be saved and which to be left out
- Input Gates
    - Updates the cell state
    - Values are transformed by a sigmoid function to be either 0(not important) or 1(important)
    - Values are also squished between -1 to 1 using a tanh function in order to help regulate the network
- Output Gates
    - Decides what the next hidden state should be
    - Passes previous hidden state and current input into a sigmoid function, then the new cell state though a tanh function
    - Multiply tanh and sigmoid outputs to decide what information to be carried by the hidden state

Convolutional LSTM (ConvLSTM)

Although vanilla LSTMs have been proven to be a very powerful model for handling temporal correlation tasks, there is too much redundancy for spatial data. There is a modification to LSTMs called the convolutional LSTM that has convolutional structures in both input-to-state and state-to-state transitions. In ConvLSTM, ConvLSTM layers are used, forming an encoding-forecasting structure that's great for spatiotemporal sequence forecasting problems. ConvLSTM models are used for analyzing spatio-temporal data and the convolutions of the

CNN are performed as part of the LSTM. The convolutions are used directly during the process of reading the input data sequences.

Shi et al explored a Convolutional LSTM network and compared it with the fully connected LSTM (FC-LSTM) model. They created this ConvLSTM layer to solve the spatiotemporal sequence forecasting or precipitation nowcasting problem. They applied the two models on a synthetic Moving-MNIST dataset and found that the ConvLSTM performed better than the FC-LSTM for their spatiotemporal sequence forecasting problem. Their ConvLSTM also outperformed the ROVER algorithm on the Radar Echo dataset.

The ConvLSTM2D class from theKeras library is used for 2D data for ConvLSTM models, but can be reconfigured to accept data in 1D for multivariate time series classification. The ConvLSTM2D class expects an input shape of size (samples, time, rows, columns, channels), where samples = the number of windows in the dataset, time = the number of subsequences that each window is split into, rows = the dimension of each subsequence, columns = the number of timesteps in each subsequent, and channels = the number of input variables.

Stacked LSTM

In the history of neural networks, many types of networks have seen improved performance by increasing depth, leading to the rise of deep neural networks. Depth is added to the network by adding intermediate nonlinear hidden layers between inputs and outputs, for example by adding many alternating convolution and pooling layers in a CNN. Increasing depth in the network allows deeper layers to further abstract information and learn higher-level features. For example, early layers in a CNN may extract edges while deeper layers extract fine-grained information specific to the image. This same concept has been explored for LSTMs as well to see if increasing depth leads to better performance.

Pescanu et al explored different types of depth in an RNN and originally coined the term “Stacked LSTM” in 2014. They explored how depth exists in an RNN because each timestep can be seen as a layer in the network, but these connections are shallow in that “there exists no intermediate, nonlinear hidden layers” between timesteps. In order to introduce true depth into the network, they explored stacking multiple recurrent hidden layers on top of each other. In this model, LSTM layers are ordered sequentially, where each previous layer passes a sequence of values, one output per input timestep, to the next layer. This introduces intermediate nonlinear hidden layers between each recurrent unit so the network can focus on the data at different timescales.

This method was explored by Graves, Mohamed, and Hinton in 2013 when they applied it to speech recognition. Graves et al experimented with 1-5 stacked LSTM layers, 250-622 hidden units per LSTM, training a little over a 100 epochs with an Adam Optimizer on recorded, sequential voice input. They discovered that performance improved by adding more layers but degraded after the network was deeper than 5 layers and that network depth had more effect on performance than layer size (number of hidden units used).

Table 5: Graves et al Stacked LSTM Performance on TIMIT Recognition

Table 1. TIMIT Phoneme Recognition Results. 'Epochs' is the number of passes through the training set before convergence. 'PER' is the phoneme error rate on the core test set.

NETWORK	WEIGHTS	EPOCHS	PER
CTC-3L-500H-TANH	3.7M	107	37.6%
CTC-1L-250H	0.8M	82	23.9%
CTC-1L-622H	3.8M	87	23.0%
CTC-2L-250H	2.3M	55	21.0%
CTC-3L-421H-UNI	3.8M	115	19.6%
CTC-3L-250H	3.8M	124	18.6%
CTC-5L-250H	6.8M	150	18.4%
TRANS-3L-250H	4.3M	112	18.3%
PRETRANS-3L-250H	**4.3M**	**144**	**17.7%**

Bidirectional LSTM

Bidirectional LSTMs analyze an input sequence from both directions - going forward and backwards in time, to make better predictions at the current time step. This may seem counterintuitive at first but imagine a movie review. Being able to see what the next 10 words in the review may help you better determine if the person is content or discontent with the movie. Expanding this example to our time series activity data, imagine we are just analyzing acceleration in the x-dimension. Looking at the last 10 readings, we may see acceleration is increasing and can use this information to guess the participant is walking forward. But the next 10 readings show decreasing acceleration, so it turns out that the participant was actually about to sit down. If our network is able to look ahead a few readings into the future, it can make better predictions about what activity the participant is currently performing. This bidirectional dependency exists in our time series data because future sensor readings can inform present ones in the same way that past readings do.

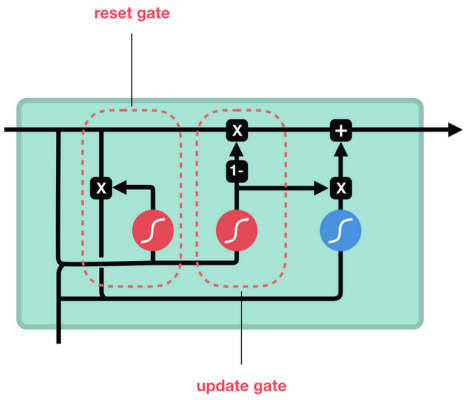
Bidirectional properties on human activity data was explored by both Hernandez et al and Yu et al where they were able to achieve almost 93% average accuracy in classifying the six types of activities in the dataset. Bidirectional LSTMs are often combined with deep, stacked LSTMs to further improve performance.

GRU

GRUs are a variation of the LSTM, in that it does similar things such as deciding what important information to keep and what to leave out. The main difference between the two is that there are only two gates in the GRU instead of the three gates in the LSTM. The gates that are present in the GRUs are the reset gate and the update gate. Since GRUs do the same thing as LSTMs, why are GRUs getting more and more popular recently over using LSTMs? This is because GRUs are much less complex than LSTMs. With only two gates and no cell state, GRUs are more computational efficient than its counterpart. GRU controls the flow of information like the LSTM but without having to use a memory unit. Exposing the full hidden content without any control, GRUs train faster usually and are much better when performing on less training data,

but in theory, pales in comparison to LSTMs when dealing with longer sequences of data. And just like LSTMs, the vanishing gradient problem is eliminated using GRUs. Below is an illustration of the GRU model and the description of its gates:

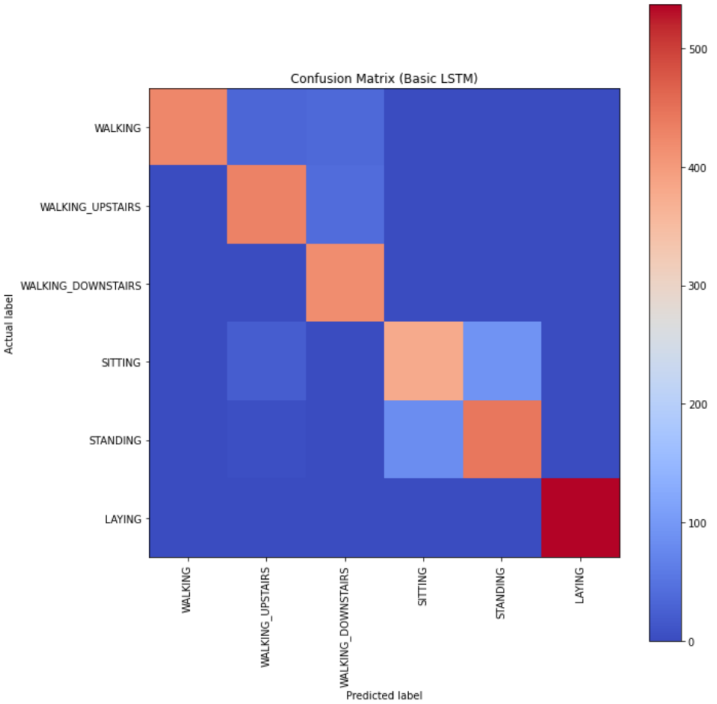
- Reset Gates
  - Decides how much past information to forget
- Update Gates
  - Similar to the forget gate in LSTMs
  - Decides how much of the past information from previous timesteps needs to be passed along to the future



Results

Basic LSTM

The basic LSTM performed surprisingly well for a model with a simple architecture and only one hidden layer of LSTM units. This architecture includes one LSTM layer, followed by one dropout layer, and two dense layers. The hyperparameters that yielded the best results are: 256 epochs, 100 units for the LSTM layer, and 0.5 Dropout. The average accuracy out of 10 trials was 89.606% with a standard deviation of 1.240. As seen in the confusion matrix, the model performed the best in classifying the laying action but didn't perform as well in classifying the sitting and standing actions.



	Basic LSTM
1	86.868
2	88.972
3	90.193
4	88.225
5	90.397
6	91.449
7	90.092
8	90.397
9	90.126
10	89.345
min	86.868
max	91.449
avg	89.606
std dev	1.240

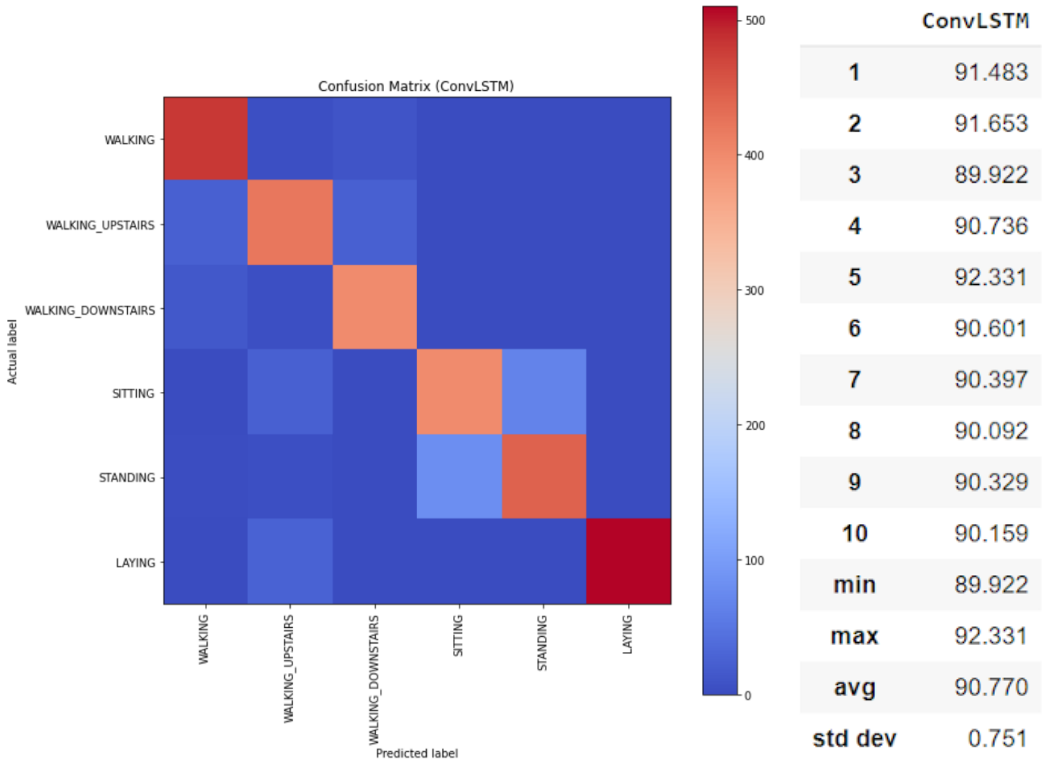
Figure 8: Confusion Matrix and Accuracies of Basic LSTM

```
basic LSTM model
Model: "sequential_9"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm_9 (LSTM)                (None, 100)               44000
_________________________________________________________________
dropout_9 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_18 (Dense)             (None, 100)               10100
_________________________________________________________________
dense_19 (Dense)             (None, 7)                 707
=================================================================
Total params: 54,807
Trainable params: 54,807
Non-trainable params: 0
_________________________________________________________________
```

Figure 9: Architecture of the Basic LSTM

Convolutional LSTM

The ConvLSTM model performed slightly better than the basic LSTM model. The architecture is similar to that of the basic LSTM model, replacing the LSTM layer with a ConvLSTM2D layer and adding a Flatten layer after the Dropout layer. Hyperparameters that yielded the best results are: 256 epochs, 64 filters and 2D kernel for the ConvLSTM2D layer, and 0.5 Dropout. The average accuracy out of 10 trials was 90.770% with a standard deviation of 0.751. This ConvLSTM model performed better overall compared to the basic LSTM as you can see from the standard deviation value. As seen in the confusion matrix, the ConvLSTM model also performed well in classifying the laying action and performed better than the basic LSTM in classifying the walking action. Similar to the basic LSTM, the ConvLSTM didn't perform as well in classifying the sitting and standing actions.



	ConvLSTM
1	91.483
2	91.653
3	89.922
4	90.736
5	92.331
6	90.601
7	90.397
8	90.092
9	90.329
10	90.159
min	89.922
max	92.331
avg	90.770
std dev	0.751

Figure 10: Confusion Matrix and Accuracies of ConvLSTM

```
ConvLSTM model
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv_lst_m2d_9 (ConvLSTM2D)  (None, 1, 30, 64)         56320
_________________________________________________________________
dropout_19 (Dropout)         (None, 1, 30, 64)         0
_________________________________________________________________
flatten_9 (Flatten)          (None, 1920)              0
_________________________________________________________________
dense_38 (Dense)             (None, 100)               192100
_________________________________________________________________
dense_39 (Dense)             (None, 7)                 707
=================================================================
Total params: 249,127
Trainable params: 249,127
Non-trainable params: 0
_________________________________________________________________
```

Figure 11: Architecture of the ConvLSTM

LSTM Hyperparameters

To further improve performance on the base LSTM our team explored tuning various hyperparameters such as number of hidden units in the LSTM, epochs, and dropout level. LSTMs use a hidden state to track sequential dependencies in the input, our model uses dropout to prevent overfitting, and number of epochs is used to determine how long to train the model for. Models were trained on Google Colab with GPU support where the average epoch took about 1 second, so testing over a wide range of epochs was inexpensive, taking at most 10 minutes. We tested average accuracy over various hyperparameter combinations to find that 150 hidden units, trained over 512 epochs with a 75% dropout level produced the best results.

Table 6: LSTM Average Accuracy by Number of Hidden Units

	50 units	100 units	150 units
64 epochs	85.51	84.05	85.48
128 epochs	85.41	89.14	87.99

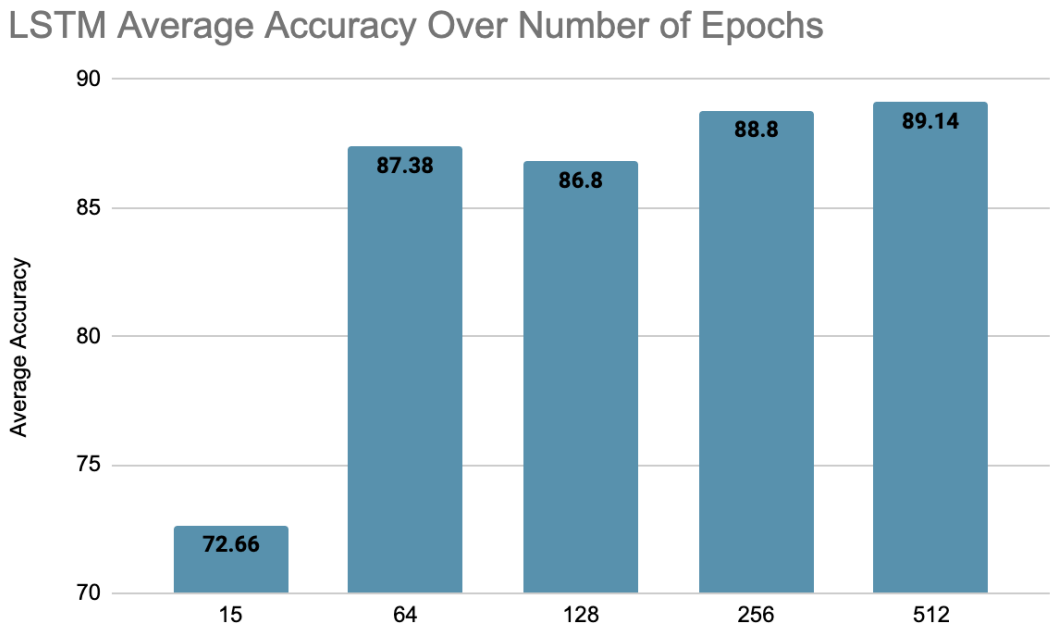


Figure 12: LSTM Average Accuracy Over Number of Epochs

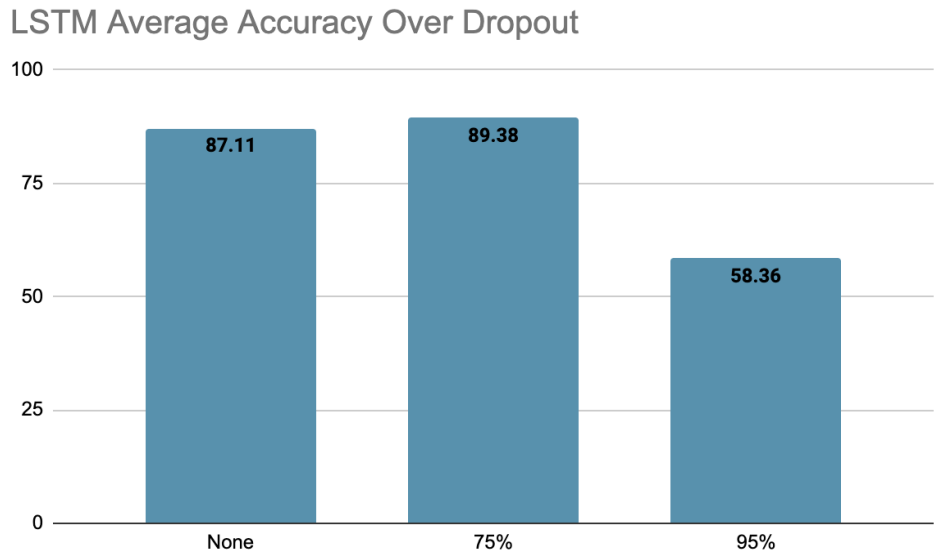


Figure 13: LSTM Average Accuracy vs. Dropout Level

Stacked LSTM

Our Stacked LSTM architecture was similar to that of Ullah et al's, applying a dense layer with ReLu activation to pre-process data before feeding the input sequence to various LSTM layers, with a final fully connected and softmax layer to produce predictions. The first dense layer is meant to normalize the signals before the LSTM, as recommended in Ullah et al s work. The input sequence to the first LSTM is a 128x100 vector instead of a 128x9, because of the initial activation layer. We experimented with a different number of hidden units and received similar results but were not able to reproduce the 93% accuracy cited in the paper. We experimented with 3-5 layers, getting the best average accuracy of 89.07% with five layers.

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_41 (Dense)             (None, 128, 100)          1000
_________________________________________________________________
lstm_44 (LSTM)               (None, 128, 150)          150600
_________________________________________________________________
lstm_45 (LSTM)               (None, 128, 150)          180600
_________________________________________________________________
lstm_46 (LSTM)               (None, 128, 150)          180600
_________________________________________________________________
lstm_47 (LSTM)               (None, 150)               180600
_________________________________________________________________
dropout_19 (Dropout)         (None, 150)               0
_________________________________________________________________
dense_42 (Dense)             (None, 100)               15100
_________________________________________________________________
dense_43 (Dense)             (None, 6)                 606
=================================================================
Total params: 709,106
Trainable params: 709,106
Non-trainable params: 0
_________________________________________________________________
```

Figure 14: 4-Layer Stacked LSTM Architecture

Similar to Ullah's and Hernandez's work we found our model had the most difficulty classifying sitting and standing activities.

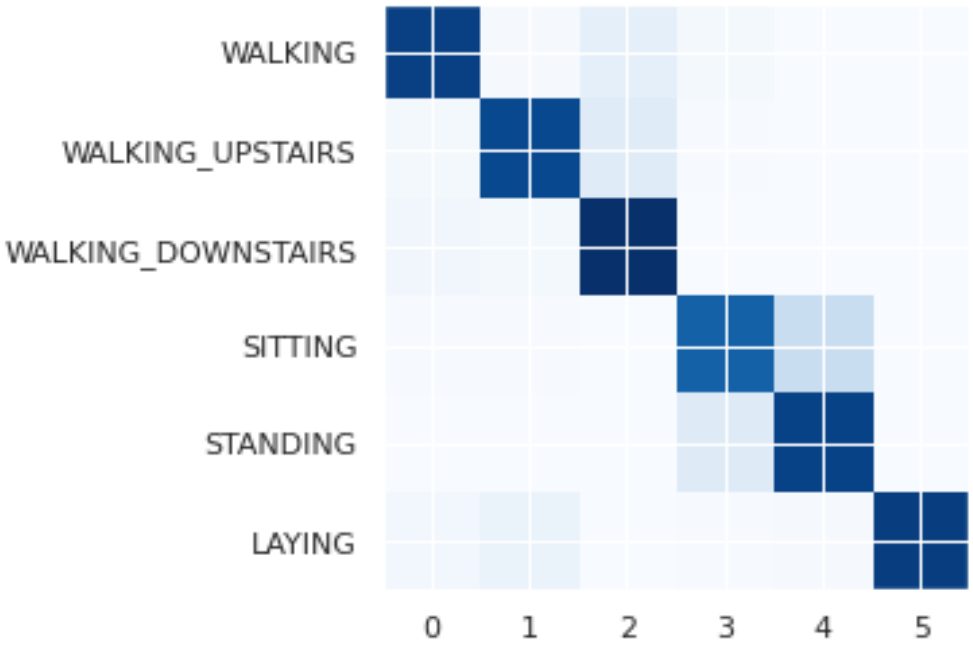


Figure 15: 5-Layer Stacked LSTM Confusion Matrix

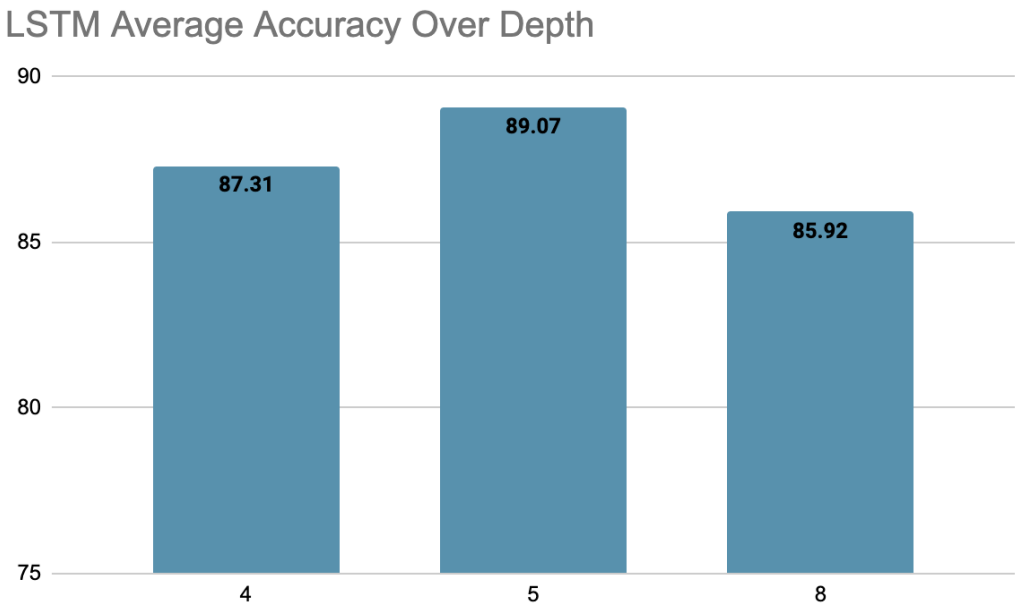


Figure 16: Average LSTM Accuracy By Depth

Bidirectional LSTM

In addition to experimenting with multiple LSTM layers, our team also experimented with bidirectional LSTMs. As explained above, bidirectional LSTMs can look at future inputs in the sequence to help inform present decisions. We decided to explore these relationships with an added bidirectional layer to our deep LSTMs. Our bidirectional model contained between 1-3 LSTM layers with bidirectional dependencies, a dropout layer, one fully connected layer, and a final softmax layer to produce predictions. Training over 512 epochs, we found about the same average accuracy with 1 layer (87.04%) vs. 3 layers (87.48%). We thought that analyzing the input sequence from both directions would help improve performance but found slightly lower accuracy with this network.

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectional (None, 300)               192000
_________________________________________________________________
dropout_20 (Dropout)         (None, 300)               0
_________________________________________________________________
dense_44 (Dense)             (None, 100)               30100
_________________________________________________________________
dense_45 (Dense)             (None, 6)                 606
=================================================================
Total params: 222,706
Trainable params: 222,706
Non-trainable params: 0
```

Figure 17: 1-Layer Bidirectional LSTM Architecture

Similar to our other LSTMs the model had difficulty classifying sitting and standing activities.

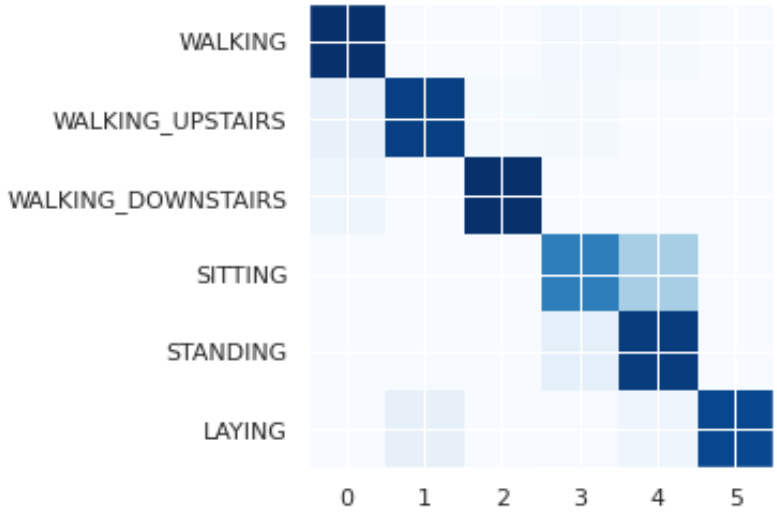


Figure 18: Bidirectional LSTM Confusion Matrix

GRU

The parameters used for this model in our project are 300 epochs, 1500 batch size, a learning rate of 0.0025 and a lambda of 0.0015. Our model also includes 2 GRU cells and 1 multi rnn cell for the hidden later, and 1 static rnn cell for the output layer. More GRU cells had a negative impact on the accuracies of the model. The L2 loss function is used to calculate the loss while training. AdamOptimizer was used as the optimizer for the model. This model managed to get an average accuracy of 89.67% across 10 runs with a standard deviation of 1.31. Similar to the previous models, the GRU had a harder time differentiating between the subjects sitting down and standing up.

	1	2	3	4	5	6	7	8	9	10	avg	Std div
Accuracy	89.45	90.90	90.90	87.24	90.60	87.92	90.16	90.26	89.34	90.92	89.67	1.31

Figure 19: Accuracies for the GRU model

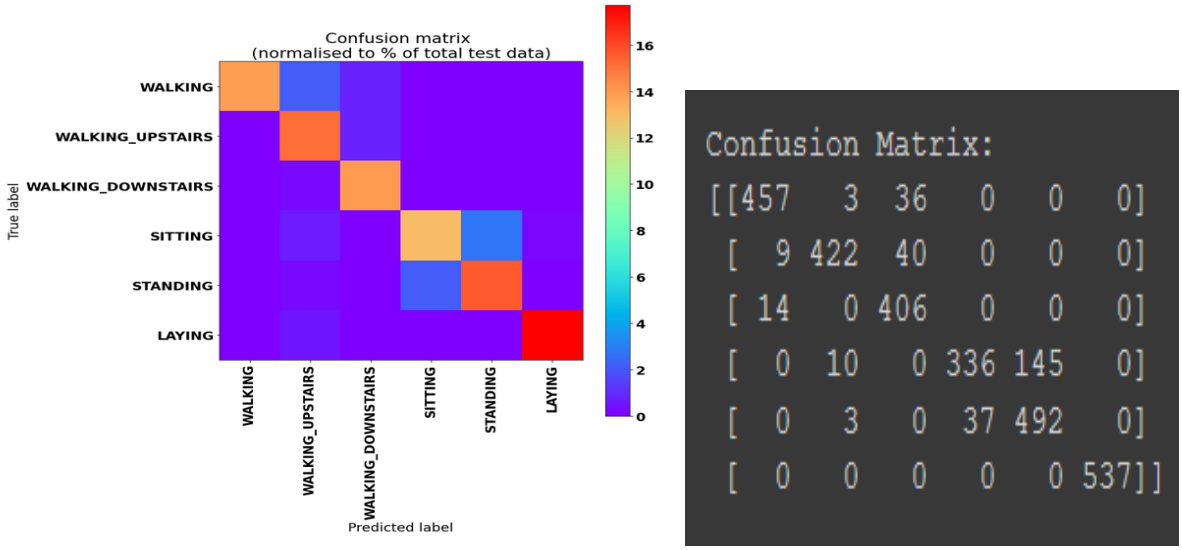


Figure 20: Confusion Matrices for the GRU model

Conclusion

In this project, we have successfully applied various types of Recurrent neural network models in order to perform classification of the six different activities from the UCI Human Activity Recognition dataset.

Overall, the accuracy metrics are similar for each model, with some performing slightly better than the others, and generally had accuracy values of around 90%. Out of the four models that we analyzed, the Convolutional LSTM model performed the best and the BiRNN model performed the worst. As seen from the confusion matrices, the different models that we analyzed performed the best for classifying the laying action, and the actions that they had worse classification accuracies were for sitting and standing. We believe that this is because the dataset contains accelerations of the subjects, so sitting and standing is more difficult to interpret and differentiate for our models since the two actions are both stationary. The subjects also had their posture in an upright position, which may have contributed to the poorer performance of our models in these aspects as well.

Citations

1. Chetty, Girija, Matthew White, and Farnaz Akther. "Smartphone based data mining for human activity recognition." Procedia Computer Science 46 (2015): 1181-1187.
2. M. Ullah, H. Ullah, S. D. Khan and F. A. Cheikh, "Stacked Lstm Network for Human Activity Recognition Using Smartphone Data," 2019 8th European Workshop on Visual Information Processing (EUVIP), Roma, Italy, 2019, pp. 175-180, doi: 10.1109/EUVIP47703.2019.8946180.
3. F. Hernández, L. F. Suárez, J. Villamizar and M. Altuve, "Human Activity Recognition on Smartphones Using a Bidirectional LSTM Network," 2019 XXII Symposium on Image,

Signal Processing and Artificial Vision (STSIVA), Bucaramanga, Colombia, 2019, pp. 1-5, doi: 10.1109/STSIVA.2019.8730249.
4. S. Yu and L. Qin, "Human Activity Recognition with Smartphone Inertial Sensors Using Bidir-LSTM Networks," 2018 3rd International Conference on Mechanical, Control and Computer Engineering (ICMCCE), Huhhot, 2018, pp. 219-224, doi: 10.1109/ICMCCE.2018.00052.
5. A. Graves, A. Mohamed and G. Hinton, "Speech recognition with deep recurrent neural networks," 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, 2013, pp. 6645-6649, doi: 10.1109/ICASSP.2013.6638947.
6. Pascanu, R., Gulcehre, C., Cho, K., & Bengio, Y. (2014). How to construct deep recurrent neural networks. In Proceedings of the Second International Conference on Learning Representations (ICLR 2014)
7. Britz, Denny. “Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs.” *WildML*, 8 July 2016, www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/. Recurrent Neural Network Diagram
8. Britz, Denny. “Recurrent Neural Networks Tutorial, Part 3 – Backpropagation Through Time and Vanishing Gradients.” *WildML*, 1 Apr. 2016, www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/. BPTT Diagram
9. Davide Anguita, Alessandro Ghio, Luca Oneto, Xavier Parra and Jorge L. Reyes-Ortiz. A Public Domain Dataset for Human Activity Recognition Using Smartphones. 21th European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning, ESANN 2013. Bruges, Belgium 24-26 April 2013.
10. Phi, Michael. “Illustrated Guide to LSTM's and GRU's: A Step by Step Explanation.” *Medium*, Towards Data Science, 28 June 2020, towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21.
11. X. Shi, Z. Chen, H. WAng, DY. Yeung, WK. Wong, WC. Woo, “Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting”,Department of Computer Science and Engineering of Hong Kong University of Science and Technology, 2015 https://papers.nips.cc/paper/2015/file/07563a3fe3bbe7e3ba84431ad9d055af-Paper.pdf