Terrain Recognition using Time-Series Data

Naga Usha Mahathi Lanka
Electrical and Computer Engineering
North Carolina State University
nlanka@ncsu.edu

Dewang Tara
Electrical and Computer Engineering
North Carolina State University
dtara@ncsu.edu

***Abstract*—This project paper proposes a deep learning approach to classify different types of terrain for human activity recognition. Time-series data of accelerometers and gyroscopes operating in x, y & z directions and corresponding class labels are used to train a deep learning model built with Bi-LSTM is used to predict the terrain type a prosthetic limb is prone to while walking. This paper presents the basic approach to augment data that is usable for a Bi-LSTM based deep learning model and further discusses the limitations in the acquired data as well the model implemented thereby outlining the details for future extension of this work. (*Abstract*)**

Keywords—Terrain, human activity recognition, Bi-LSTM, data augmentation,

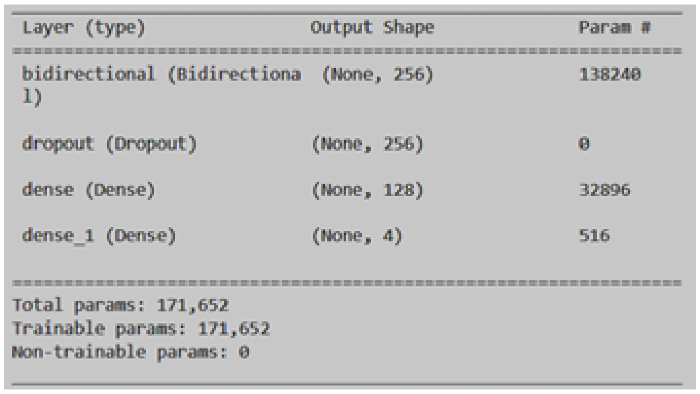
I. METHOLDOLOGY

The main objective of this project is to develop a terrain identification system using inertial measurement units (IMU) data gathered from sensors attached to the lower limb of 8 different subjects. The data consists if 3 measurements from accelerometers and 3 from gyroscopes. The output classes include standing or walking in solid ground, going down the stairs, going up the stairs, and walking on grass.

The methodology for implementation of our model was split into different categories namely data pre-processing, data splitting and model architecture. We implemented a simple Bi-LSTM based deep learning model for this problem as we have time-series data.

By thorough inspection, we figured that the sampling rate for x is 40 Hz and sampling rate for y is 10 Hz. For every 0.1s of y, 4 samples of x were available. The label from y were extrapolated and mapped to 4 rows of IMU sensor data corresponding to x. This generated a balanced data where each row of measured sensor value consisting of 6 data points (x1, y1, z1, x2, y2, z2) along with its extrapolated class label became a single data instance.

Sequence of the data instances were passed to train using the Bi-LSTM model. The advantage of using LSTM'S for sequence classification is that they can learn from the raw time series data directly and thus do not require a thorough understanding of the input features. Functions native to Keras were employed for this project, including model generator functions, layer functions, the Bi-LSTM function and the early stopping function which were some of the most important ones.



```
Layer (type)                 Output Shape              Param #
=================================================================
bidirectional (Bidirectiona  (None, 256)               138240
l)

dropout (Dropout)            (None, 256)               0

dense (Dense)                (None, 128)               32896

dense_1 (Dense)              (None, 4)                 516

=================================================================
Total params: 171,652
Trainable params: 171,652
Non-trainable params: 0
_________________________________________________________________
```

Figure 1 Bi-LSTM model with dropout ad dense layers

The Bi-LSTM model was designed in such a way that it has a Bi-LSTM layer followed by a dropout layer to avoid overfitting. 2 dense layers were added in the end with the output being 4 units wide corresponding to each class. Refer to the above table.

II. MODEL TRAINING AND SELECTION

A. *Model Training*

In this project, even though it was a classification problem we did not implement classical machine learning models like Random Forest classifier in favor of producing a more accurate deep learning model. The general structure of the model described in the methodology section is just to build on the bi-LSTM model by adding a few dense layers and a dropout layer to ensure that there is no overfitting. For the training data, the interpolated data was used as the input after having been split into train, validation and test data.

The model was separately trained on each user session and the data from each user session is fed in batch sizes of 128 and 10 epoch each. 10 epochs were chosen because when the model was trained with more than 10 epochs, the difference in between validation loss and train loss kept increasing there by increasing the error.

The input data for training was sampled at a rate that was four times faster than the output data. To accommodate for this, the output data was augmented by extrapolating each output four times to get a one-to-one relationship on the data for training.

B. *Model selection*

After performing the training on each user session, the hyperparameters for the model were selected based on the performance of the model. An increase in epochs was beneficial for the single session, but when the training was increased to train on every user session, more than 10 epochs were too many, and the validation accuracy started diverging.

Parameter	**Value**
Epochs	10
Batch Size	128
Dropout rate	0.5
Learning rate	0.001
Optimizer	Adam
Loss function	Cross entropy

Figure 2 Static Hyperparameters used in the model architecture

All the above mentioned hyperparameters were maintained constant throughout the model. The following hyperparameters were used to tune the model performance.

a) Window Size: The size of the window determines the time taken by the prosthethic limb to trave on a particular terrain. We decided to use a window size of 30 on the data from x which is equivalent to 0.75s and also a window size of 60 which is equivalent to 1.5s.

Window size	Accuracy	F1 Score
60	0.94	0.92
30	0.92	0.89

b) Hidden Units: The number of hidden units determine the capability of the model to generate various features based on the data.

Hidden units	Accuracy	F1 score
128	0.94	0.93
256	0.91	0.88

c) Step size: Step size is an important parameter in determining how much the model should skip in batch while training. This will help in elimminating some of the data which can create anomaly in the dataset. (Basically helps in discarding the outliers)

Step size	Accuracy	F1 score
1	0.95	0.93
4	0.92	0.90

After careful inspection, based on whichever hyperparameters yielded higher F1 scores and accuracies, Bi-LSTM model was chosen as our model as it can use temporal information in both directions for making predictions.
We observed a trend in loss per epoch that increased as the number of epochs were increasing during the training time. This is the best model that we achieved. To play with this network more, we could try experimenting on different dropout rates and different numbers of LSTM layers and hidden units in them.

III. Evaluation

In order to evaluate the performance of this network, six main evaluation metrics were considered: training error, validation error, precision, recall, accuracy, F1 scores. With these six metrics it is possible to develop a reasonable understanding of how the system has performed.

After training the bidirectional LSTM network for 10 epochs, this system was found to have a validation accuracy around 94.49% and a validation loss of approximately 0.288. The training accuracy for this system was found to be around 98% with a training loss of around 0.016 which clearly shows the overfitting that is happening in the training dataset. Even after using early stopping to prevent overfitting, This large difference in accuracy between the training data and validation data is most likely due to an imbalance in the validation data and should be accounted for in further versions of this project.

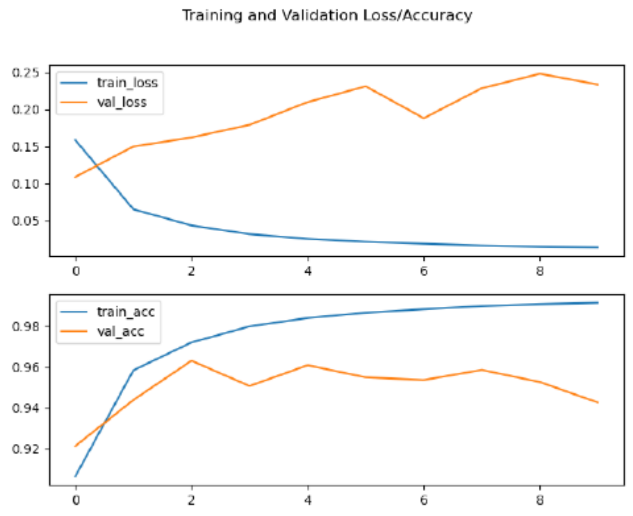
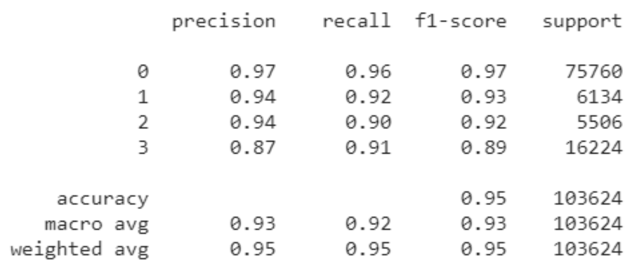


Figure 3 Validation and training losses and accuracies for 10 epochs

Predictions were made from the resultant model which outputs a 1×4 array of probabilities for each category for each timestamp. This data was transformed into a 1×n matrix that list the overall prediction per timestamp given the probability matrix, where n is the number of timestamps sampled in the prediction. Predictions were made for subjects 9-12.

The precision, recall and F1 score were calculated for a small window and had the following results. Precision: 0.93, accuracy: 0.92, F1 score: 0.93.


```
              precision    recall  f1-score   support

           0       0.97      0.96      0.97     75760
           1       0.94      0.92      0.93      6134
           2       0.94      0.90      0.92      5506
           3       0.87      0.91      0.89     16224

    accuracy                           0.95    103624
   macro avg       0.93      0.92      0.93    103624
weighted avg       0.95      0.95      0.95    103624
```

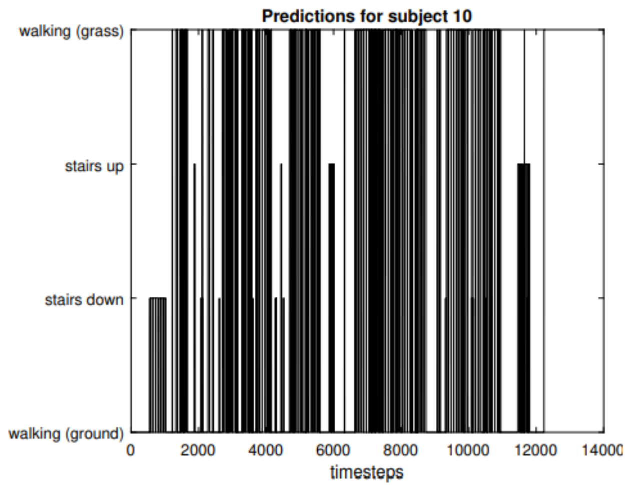


Figure 4 Subject 10 prediction values

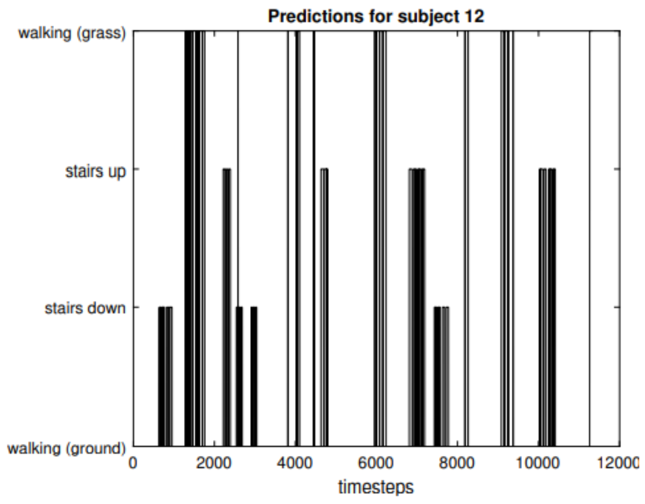


Figure 5 Subject 12 prediction values

The predictions show reasonable movement with each subject within the duration of their individual test.

References

[1] Jinna Kim, Nammee Moon, "Bi-LSTM model based on multi-variate time series data in multiple field for forecasting trading area," Journal of Ambient Intelligence and Humanized Computing, DOI: 10.1007/S12652-019-01398-9

[2] J. Brownlee, "How to Calculate Precision, Recall, F1, and More for Deep Learning Models," Machine Learning Mastery, 27-Aug-2020. [Online]. Available:https://machinelearningmastery.com/how-to-calculate precision-recall-f1-and-more-for-deep-learning-models/.

[3] A. Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks, 21-May-2015. [Online]. Available: http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[4] J. Brownlee, "How to One Hot Encode Sequence Data in Python," Machine Learning Mastery, 14-Aug-2019. [Online]. Available: https://machinelearningmastery.com/how-to-one-hot-encode-sequence data-in-python/. K. Elissa, "Title of paper if known," unpublished.

[5] J. Brownlee, "How to Reshape Input Data for Long Short-Term Memory Networks in Keras," Machine Learning Mastery, 14-Aug-2019. [Online]. Available: https://machinelearningmastery.com/reshape-input-data-longshort-term-memory-networks-keras/

[6] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional LSTM and other neural network architectures," Neural Networks, vol. 18, no. 5-6, pp. 602–610, 2005. M. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.