[Team 32] Terrain Identification from Time Series Data using class-weighted LSTM

CHAITANYA RAJEEV
Email: crajeev@ncsu.edu

SAI SASHANK NAYABU
Email: snayabu@ncsu.edu

LAXMI AISHWARYA THELA
Email: lthela@ncsu.edu

I. Methodology

The terrain identification problem consists of predicting from among 5 different surfaces on which a prosthetic leg is currently walking on, by analysing data from accelerometers and gyroscopes placed on the device. Accelerometers measure linear acceleration (specified in mV/g) along X, Y, Z axes. A gyroscope measures angular velocity (specified in mV/deg/s) along the same axes. Both gyroscope and accelerometer are connected to the knee of the subjects. Data is generated through making each subject (8 in total) walk on for a few seconds on a combination of terrains. The different terrain classes walked on were grass, solid ground, marble, tile, upstairs, and downstairs.

Since the data can be classified as a time-series, an LSTM neural network was built for prediction. LSTMs are a class of Recurrent Neural Network architectures (RNN) that have the unique ability to tune weights that allow them to hold memory from previously seen data. This allows them to perform especially well on data with temporal components.

A) Dataset Description

The datasets consist of 7 predictors (3 each for accelerometer axes and gyroscope axes) and a time stamp when the readings were recorded. There was a total of 8 subjects. Certain subjects had multiple ‘walks’ of recorded data. The response variable, which was the numerically encoded terrain class, was provided in a separate series of datasets each corresponding to their respective ‘walk’ and subject.

The sensor readings were sampled at 40 Hz whereas the output terrain class was sampled at 10 Hz. This effectively meant that there were 40 sensor data points and 10 terrain output classes for 1 second of walking. Due to the sampling imbalance, every sensor dataset had 4 times the data points as compared to the corresponding response dataset.

B) Data Pre-processing

The first step in data pre-processing was up sampling the response data to have one response observation for every datapoint in the corresponding sensor dataset. To accomplish this, we used a KNN classification algorithm along the time-step domain to predict an output class for every row of sensor data.

Briefly described, the algorithm takes the time value of the first row in the sensor dataset, finds out the 5 nearest time values in the response dataset and takes a majority vote (mode) of the corresponding 5 classes. The majority class is then assigned to the first row of the sensor data as the response for that row. This procedure is repeated for every row in the sensor dataset.

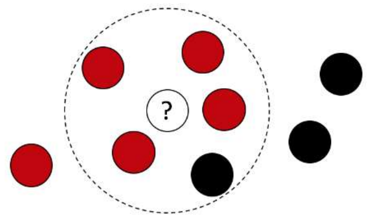


Figure 1: KNN classification illustration in 2D

Once response data was generated for all data points, the datasets were combined into a single dataset for training. The pandas and sci-kit learn library were widely utilized for these operations.

The next step was to divide the data into a training set and a validation set. Since the data was time-series in nature, a randomized split would corrupt the ordering of the data. Hence, the walks of the first 6 subjects were taken to be the training data, and that of the 7th and 8th subject was the validation set.

C) Model Architecture

The LSTM architecture that we used was simple and consisted of one recurrent layer of 100 LSTM cells followed by a Dropout layer to prevent overfitting. This was followed by a fully dense connected layer of 100 neurons, one for each LSTM cell. This was then followed by another dense layer of 4 neurons, one for each class of the response occurring within our training set.

ReLU activations were used the first dense layer due to its desirable gradient properties including faster training and its resistance to vanishing/exploding gradients issues. A Softmax activation was used for the second dense layer to map activations to relative occurrence probabilities. The prediction was then a binary 4-dimensional vector which could then be transformed to a value among 0,1,2 and 3.

Adam was chosen as the optimizer as it is widely used and yielded the best performance while tuning hyper-parameters. The final model has 53,304 trainable parameters put together. The loss function was chosen to be categorical cross-entropy and the accuracy metric was categorical accuracy. We used the Keras library to build out and train the entire architecture.

II. Model training and selection

In this section, we shall discuss the pre-processing steps taken to feed data into the model for training and also the methods that were used to handle class imbalance. We shall also discuss the hyper-parameter tuning process and present relevant plots.

A) *Model Training*

1. *Data transformation for training*

In order to train the LSTM, the data had to be morphed into a format that is accepted by a Keras LSTM layer. To achieve this, we implemented a moving window style procedure which takes a window of data points and then transforms it into a matrix within a tensor.

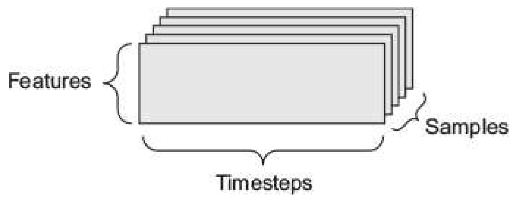


Figure 2: LSTM input tensor format

An illustration of the input tensor is shown in Figure 2. The size of the window is the timesteps dimension. We chose 25 data points as our timesteps dimension or window size. The step size or the stride of data points the time windows moves for each iteration was chosen to be 4 because the final predictions had to again be down sampled from 40 Hz to 10 Hz. The batch size which is also the no. of samples in one pass was chosen to be 64. Finally, we had 6 features (sensor predictors). Our final input tensor was of the dimension format (no. of samples, time steps, features).

2. *Handling class imbalance*

The training dataset came with significant data imbalances. About 75% of the responses were of class 0, 15% for class 3, followed by 4% and 5% for classes 1 and 2 respectively. Training the algorithm on the current dataset will cause the algorithm to perform poorly on minority class which will negatively impact the f1 scores.

To address this, we passed class weights to the cost function in order to prioritize minority classes. The weights were inversely proportional to the frequency of occurrence in each class. For class 'i', the weight is calculated as:

$$weight_i = \frac{Total\ no. of\ samples}{no. of\ unique\ \ classes \times no. of\ class\ 'i'\ samples}$$

This formula was borrowed from the sci-kit learn documentation for balanced class weights function argument. The following table summarizes the calculated weights for each class using our training dataset:

Class	**0**	**1**	**2**	**3**
Weight	0.33	5.56	4.45	1.74

B) *Model Selection*

1. *Optimizer Selection*

Before performing hyper-parameter tuning to select the best dropout rate and learning rates, we first compared three different optimizers i.e. SGD, RMSprop and Adam and their learning curves to determine the best optimizer. Results are plotted over 10 epochs.

From the above plots, it is

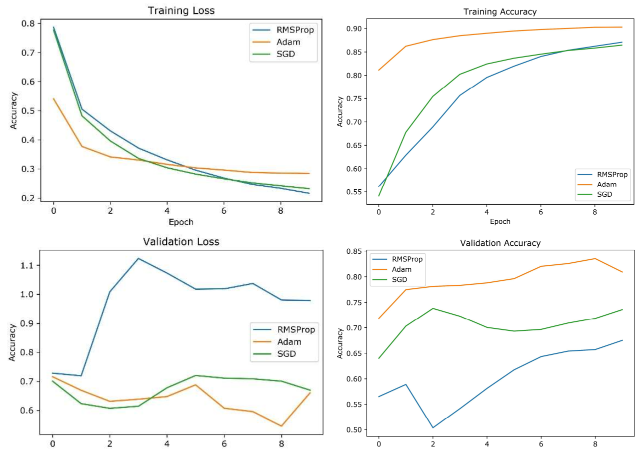


Figure 3: Training/Validation accuracy/loss curves for different optimizers

clear that the Adam optimizer is the best choice as it gives the lowest validation loss and highest validation accuracy. Note that due to the small size of the validation set, the validation accuracies show high variances and fluctuations.

2. *Hyper-parameter tuning*

Now that the Adam optimizer has been chosen, the next step is to tune the dropout rate and learning rate to arrive at the best combination of these parameters. 0.1, 0.2 and 0.4 were the candidates for dropout rate whereas 0.0005, 0.001 and 0.005 were the candidates for the learning rate.

From the above plots, the best combination of parameters

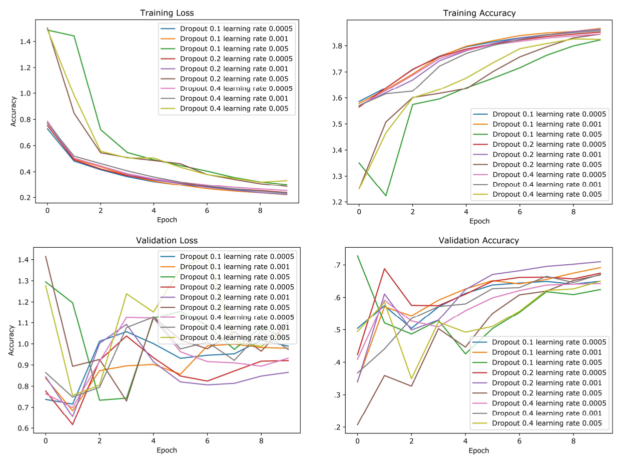


Figure 4: Training/Validation accuracy/loss curves for various dropout/learning rates

which produces the lowest validation error and highest validation accuracy is for a Dropout Rate of 0.2 and a Learning Rate of 0.001. So these along with the Adam optimizer are the chosen values for our hyper-parameters.

III. Evaluation

In the final section we shall discuss the performance of our model with the chosen hyperparameters (Dropout rate: 0.2, learning rate: 0.001, Optimizer: Adam) on the validation data. We shall also present prediction visualization plots for selected time snippets for illustration.

For the final submission, we combined the validation and training data into one dataset and trained the final model on the entire data in order to maximize the training sample size for the best possible submission f1 score.

A) Performance on the Validation Set

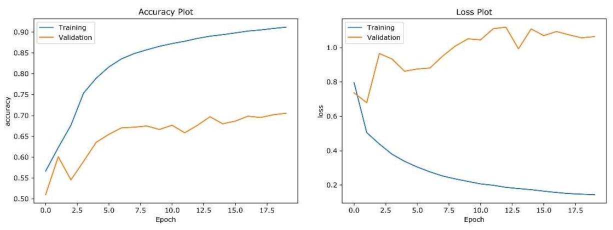


Figure 5: Accuracy/Loss plots of the final model

The validation loss of the model seems to be increasing with each passing Epoch. This might seem surprising at first. But we must remember that we have previously weighed the cost function to prioritize training to predict minority classes better. Hence this might increase miss-classifications in the majority class causing the loss value to go up. From the training plot, the validation accuracy after 20 epochs touched 70%.

The confusion matrix of the validation set predictions is displayed below.

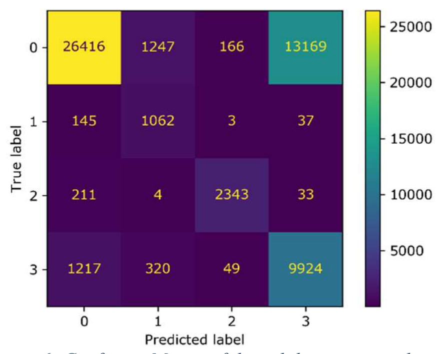
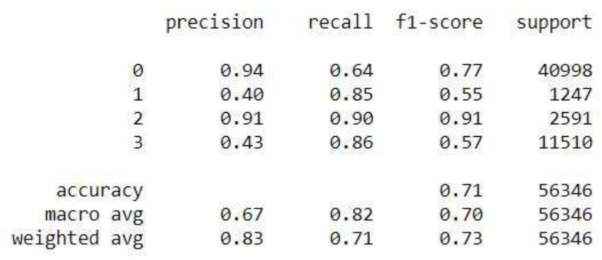


Figure 6: Confusion Matrix of the validation set predictions.

In Figure 6, a full classification report detailing multiple metrics like precision, recall, accuracy and f1 score (harmonic mean of precision and recall) are provided.



```
              precision    recall  f1-score   support

           0       0.94      0.64      0.77     40998
           1       0.40      0.85      0.55      1247
           2       0.91      0.90      0.91      2591
           3       0.43      0.86      0.57     11510

    accuracy                           0.71     56346
   macro avg       0.67      0.82      0.70     56346
weighted avg       0.83      0.71      0.73     56346
```

Figure 7: Classification report of validation set.

From the confusion matrix, there seems to be significant misclassification of class 0 as class 3. This is also confirmed by the classification report where f1 scores of class 1 and class 3 are low.

In order to correct this, the model class weights for the cost function were tweaked, and the model was trained for further epochs for the final test predictions.

B) Performance on Test Data (Gradescope)

For the final prediction, the entire undivided training data was used to train the model for 20 epochs. The chosen parameters were as follows:

Dropout	Learning Rate	Optimizer	Batch size
0.2	0.001	Adam	64

Class weights used were:

Class	0	1	2	3
Weight	0.33	6.07	4.59	1.62

With these parameters, the final model achieved an f1 score of 0.834 on the test data in Gradescope.

C) *Illustration of test set predictions*

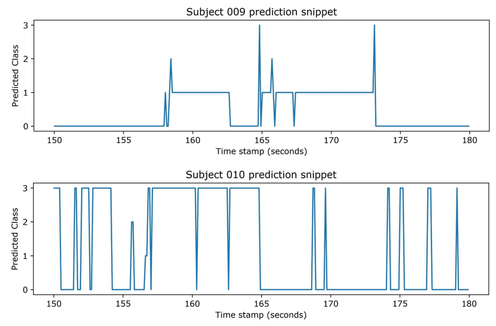


Figure 8: Prediction visualization plots (example snippets)

Figure 8 shows prediction plots for random snippets for test subjects 009 and 010. The X-Axis shows the timestamp in seconds. Both snippets show the predictions between 150 and 180 seconds into the respective walks. From the snippet of subject 010, it looks like that subject might have been alternating between various surfaces during his/her walk.The Y-Axis represents classes 0 (solid ground), 1 (downstairs), 2 (upstairs) and 3(grass).

References

[1] https://machinelearningmastery.com/how-to-develop-rnn-models-for-human-activity-recognition-time-series-classification/

[2] Wen, Qingsong, et al. ”Time series data augmentation for deep learning: A survey.” arXiv preprint arXiv:2002.12478 (2020).

[3] I. Goodfellow, Y. Bengio, A. Courville ”Deep Learning,” MIT Press,