Towards quantitative rehabilitation of stroke patients via deep learning

Aakash Kaku ARK576@NYU.EDU *Center for Data Science*
New York University
New York City, NY, US

1. Introduction

Strokes are the leading cause of disability in the United States. They affect almost 1 million patients in the United States annually, leading to a total cost of about USD 240 Billion (Go et al., 2014)(Ovbiagele et al., 2013). Almost two-thirds of stroke patients have significant impairment in their upper limbs or extremities. This impairment leads to limitations in the performance of activities of daily living (ADLs), like feeding, bathing, grooming, dressing, etc. The impairment can be reduced and mitigated with the help of a clinical intervention namely, rehabilitation training which typically involves the repeated practice of ADLs composed of basic building blocks of motion, called functional primitives.

Just like we have phonemes for speech, these primitives can be considered to be phonemes for activities. Primitives can be flexibly strung together to create an activity or an ADL. There are five fundamental primitives which combine to form any activity: *reach, idle, stabilize, transport,* and *reposition*. For example, when an individual reaches for a glass of water, the hand that reaches out to grab the glass is performing a *reach*.

The rehabilitation training started in the early weeks after the stroke helps the patients to recover quickly from the impairment. However, an unanswered question is how much rehabilitation training should the patient perform? Currently, the dose of rehabilitation training for humans depends on the intuition and experience of the administering therapist. This makes the rehabilitation training more of an art than science. The rehabilitation training is not systematically quantified in humans because it is prohibitively time-consuming to manually annotate and tally the number of functional primitives performed by the patient.

Some proxies can be used to quantify the rehabilitation training like time of the session or using videos to track motion in order to count the movements automatically. Unfortunately, time is not a good surrogate for the number of functional movements done by the patient in a particular training session, as shown by Lang et al. (2007). Using videos to track motion also has limited applicability since patients' actions can be occluded in the videos, we might need multiple viewpoints, or we may not be able to identify primitives due to noisy visual backgrounds. In this work, we try to solve this problem by using sensors attached to the patients' body, which capture signals like accelerations, rotations, etc. and identify the primitive movement done by the patient. We take the critical first step to automatically identify functional primitives using the machine learning models. We feed the sensor data which is captured using a measurement device called Inertial Measurement Units (IMUs) to machine learning models and identify the primitive performed by the patient. This way, we can quantify the rehabilitation training of a patient and help to maximize patients' recovery.

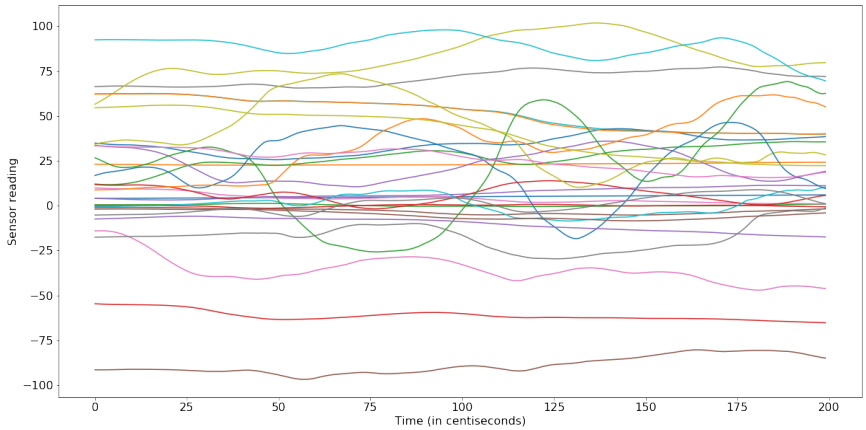


Figure 1: A sample IMU data, which is 200 time-step long (2 secs long).The plot shows several lines where each line represents how the reading of a single sensor changes over time. Such a sample IMU data which is in $\mathbb{R}^{76\times200}$ acts as an input to a machine learning model that identifies the primitive taking place in it.

We structure the work as follows: First, we look at how we identify the primitives. Secondly, we compare various models used to identify primitives. And, finally, we analyze the errors of various models and discuss future steps to correct these errors.

2. Model training and evaluation methodology

Data captured using the IMUs are high-dimensional time series data as there are 76 sensors that capture 76 different physical quantities like accelerations, rotations, angular velocities, and gyroscopic readings at each time step (each time step = 0.01 secs). A sample IMU data, which is 200 time-step long (2 secs long), can be seen in figure 1. The figure shows several lines where each line represents how the reading of a single sensor changes over time. Such a sample IMU data, which is in $\mathbb{R}^{76\times200}$, acts as an input to a machine learning model that identifies the primitive taking place in it. The model outputs five scores corresponding to five primitives, i.e., *reach, transport, reposition, idle,* and *stabilize*. The prediction of the model is the primitive with the highest score. In this section, we first discuss the evaluation metric and then discuss and compare different models for identifying primitives.

2.1. Evaluation metric

We first train a model on a train set which a subset of the our data kept separate to train the models. The model is, then, evaluated it on a held-out set that is not seen by the model during the training. In this way, we evaluate how the model performs on the unseen data. In order to quantify the model's performance, we would like to determine the accuracy of the model for identifying the primitives. However, in our case, the dataset is relatively imbalanced with the smallest primitive (*repositions*) constituting approximately 10% of the data and the largest primitive (*transports*) constituting 26% of the data. Therefore, for such an imbalanced dataset, the balanced accuracy metric (1) is appropriate to gauge the model's

performance as it gives equal importance to all primitives irrespective of their imbalance in the dataset.

$$\text{Bal. Acc} = \frac{\sum_{i=1}^{5} \frac{TP_i}{P_i}}{5} \tag{1}$$

Where, TP_i = total number of true positives for the i^{th} primitive, P_i = total number of examples for the i^{th} primitive. A prediction is a true positive when the prediction matches the ground truth.

2.2. Models for predicting primitive

To learn a model that identifies the primitive is nothing but, given a set of training data, to learn a function f that maps from $\mathbb{R}^{76\times200} \rightarrow \mathbb{R}^5$. There are infinite fs that can map from $\mathbb{R}^{76\times200} \rightarrow \mathbb{R}^5$; some are flexible, whereas others are rigid, which depends on how we parametrize the function. Flexible functions are those functions that allows a lot of interaction between the different parts of the input. For example, a higher polynomial like $f = (ax_1 + bx_2)^{100}$ has $\binom{100}{2}$ interaction terms which enables the function to easily fit any data in $\mathbb{R}^2$. Whereas, rigid functions are those functions that do not allow or barely allow the different dimensions of the input to interact among themselves. For example, a linear function like $f = (ax_1 + bx_2)$ do not have any interaction term between variables x_1 and x_2.

To further understand the concept of flexible and rigid functions and how well they generalize to unseen data, we take a simple case where our training data lies in $\mathbb{R}$, and we need to learn a function $f : \mathbb{R} \rightarrow \mathbb{R}$ that fits our data. The simple case is depicted in figure 2. As seen in figure 2, a rigid function (model), which tends to be simple like a linear function, underfits the training data. Whereas a flexible function (model), which tends to be complex like a higher-order polynomial, overfits the training data. In both the cases, it is easy to see that both the functions (models) will not perform well on the unseen data. Instead, a balance should be maintained between the two extremes, and most of the machine learning research focuses on how to strike this balance. The balance between underfitting and overfitting scenario depends on how we parametrize the function (model).

If we parametrize the function in such a way that allows many interactions between different parts of the input, then it becomes easy for the model to find spurious patterns in the data and overfit to it. Whereas, if we restrict these interactions, we control the complexity of the model and make it more simple. For example, in a linear function, for a multi-dimensional input, there are no interactions between different dimensions of the input. We, therefore, can control the complexity of function (model) by controlling the interactions that the function allows within an input.

2.2.1. Traditional models prone to underfitting

The different models present in the literature are nothing but different ways of combing the information present in the input. These models aim to reduce unwanted interactions and promote useful interactions that would, eventually, help in performing the prediction task. A class of models that are commonly used to perform the prediction task do not allow the interactions to take place over the temporal dimension of the input. Remember, our input has a temporal dimension (200-time steps) as well as sensor dimension (76 sensors).

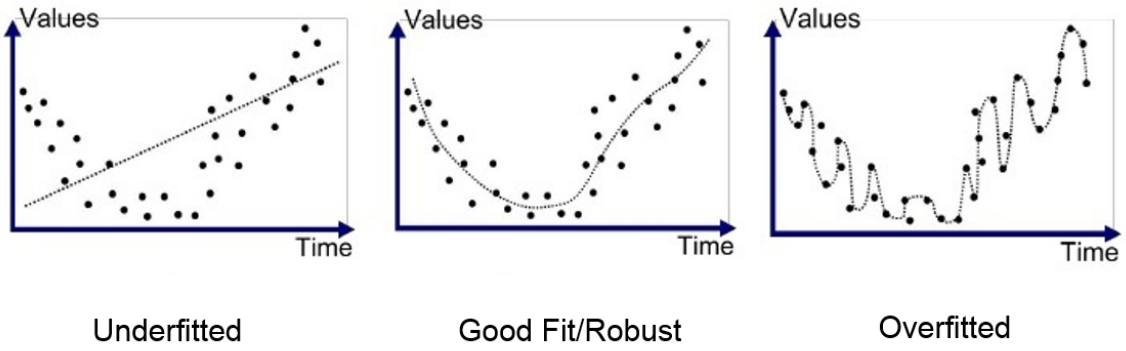


Figure 2: Left plot shows a rigid function (model) which tend to be simple like a linear function that underfits the training data. Right plot shows a flexible function (model) which tends to be more complex like a higher order polynomial that overfits the training data. In both the cases, it is easy to see that both the functions (models) won't perform well on the unseen data. Instead, a balance should be maintained between the two extremes as shown in the center plot.

The traditional models, like random forest (Breiman, 2001) or a fully connected neural network, do not allow the interactions to take place over the temporal dimension by aggregating information across it but keeping the sensor dimensionality intact. The models aggregate the information across the temporal dimension by computing statistics like mean, standard deviation, maximum, minimum, or root mean square over the temporal dimension (Kwapisz et al., 2011; Guerra et al., 2017). The model's input dimensionality, therefore, reduces from $\mathbb{R}^{76\times200}$ to $\mathbb{R}^{76\times5}$, where the fine-grained time information is lost.

This kind of dimensionality reduction controls unwanted interactions happening across the temporal dimension. However, it also throws away a lot of essential information that could be useful to perform the prediction task. This results in a scenario where the model tends to underfit because a lot of useful interactions were restricted. Results in table 1 demonstrates that the traditional models underfit the data and, hence, have a lower performance on the unseen data as compared to other models that we will discuss in the next section.

2.2.2. Deep learning models easily overfit

Deep learning models are the most complex class of functions that can learn any pattern present in the data, even spurious patterns. They, therefore, are prone to overfit to the data they are trained on. To control this overfitting, as discussed in section 2.2, the deep learning models should restrict the unwanted interactions within the input. They achieve this by only allowing some parts of the input to interact with each other. In deep learning terminology, this is called imposing a structure or an architecture. Different architectures are found to be suitable for different types of data. For example, a convolutional architecture (also known as convolutional neural network or CNN) is good for learning patterns from image data and temporal data. Whereas, a recurrent architecture (also known as recurrent neural network or RNN) is good at finding patterns in the temporal data. In this study, we select the best performing CNN and the best performing RNN (namely Long Short Term Memory network

Models	Traditional models		Deep learning models		
Model type	Random forest	Fully connected neural network	CNN	LSTM	Proposed CNN model
Balanced accuracy	52.66	58.04	64.01	66.58	**69.21**

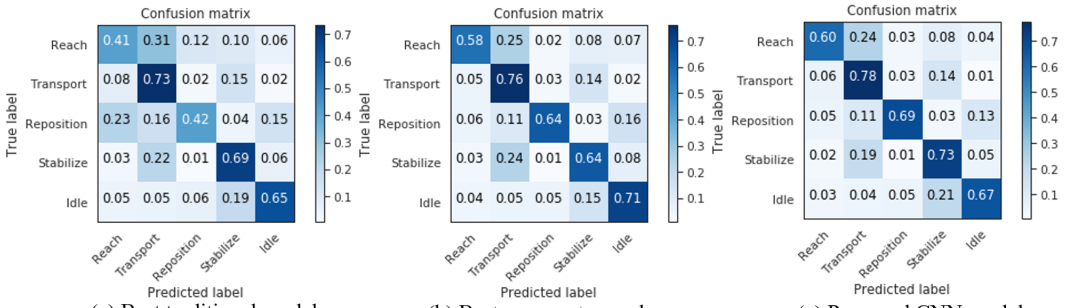
Table 1: Balanced accuracies for different models described in Section 2.2.1 and 2.2.2. Although LSTM outperforms baseline CNN model, our proposed convolutional network outperforms all the models by atleast 3-5%

or LSTM) from the literature and use it to identify primitives(Hochreiter and Schmidhuber, 1997; Wang et al., 2017). The results in table 1 show that CNN and LSTM perform better than the traditional models which suffer from the problem of underfitting.

Although these architectures tend to impose a structure and allow useful interactions within the input data, they still are suboptimal because the imposed structures have not taken into account the actual prediction task or the domain from where the data is being extracted. Therefore, the architectures need to be finetuned for our prediction task. This finetuning of the architecture requires some domain knowledge from where the data is being extracted. For example, in our case, each of the 76 sensor readings represents different physical quantities like accelerations, angular velocity, rotations, etc. A naive implementation of the best CNN, as per the literature, will allow these different physical quantities to interact freely which seems to be imprudent for our purpose as they have different measuring units. A simple change in the CNN, which restricts these interactions until the different physical quantities are mapped to a shared space, results in more meaningful interactions. As seen from the results in table 1, this simple change boosts the model's performance by 4-5% over the best CNN implemented without taking the domain knowledge into consideration.

3. Analysis of errors of the models and their limitation

To analyze the errors of the model, we plot the confusion matrix, a square matrix that shows the true labels along the rows, and the predicted labels along the columns. Each row of the matrix adds up to one, and ideally, a perfect model will have ones along the diagonal of the matrix. We plotted the confusion matrices for the best traditional model, best RNN, and our proposed CNN. All the confusion matrices demonstrated a common trend, i.e., the *reaches* were confused with *transports*, and the *idles* with *stabilizes*. In other words, all the models faced difficulty in distinguishing *reaches* from *transports* and *idles* from *stabilizes*. This trend is explained by the close resemblance of both pairs of confused primitives. They only differ by a small detail, i.e., one includes grasping the object while the other does not include grasping. However, unfortunately, the grasping information is not captured during our rehabilitation study. The models, therefore, find it difficult to differentiate between those primitives. As part of future work, we aim to extract the grasping information from the videos of the rehabilitation training and improve classification performance.



(a) Best traditional model (b) Best recurrent neural network (c) Proposed CNN model

Figure 3: Confusion matrices for (a) best traditional model, (b) best recurrent neural network, (c) proposed convolutional model. From the confusion matrices, it could be seen that *idle* is confused with *stabilize* and *reach* is confused with *transport*.

4. Conclusion

In this study, we showed that deep learning models, along with IMU sensors, could be effectively used to identify functional primitives. Such models are useful to quantify rehabilitation training in stroke patients. Additionally, we show that a simple finetuning of the existing convolutional architecture with the help of domain knowledge can significantly boost the performance of the model as compared to implementing it without using the domain knowledge. Finally, we also discussed the limitations of the model of not being able to distinguish *reaches* from *transports* and *idles* from *stabilizes*. To rectify the errors, as a future step, we aim to provide the model with additional grasping information extracted from the videos.

References

Leo Breiman. Random forests. *Mach. Learn.*, 45(1):5–32, October 2001. ISSN 0885-6125. doi: 10.1023/A:1010933404324. URL https://doi.org/10.1023/A:1010933404324.

A S Go, D Mozaffarian, V L Roger, E J Benjamin, J D Berry, M J Blaha, S Dai, E S Ford, C S Fox, S Franco, and et al. Heart disease and stroke statistics–2014 update: a report from the american heart association., Jan 2014. URL https://www.ncbi.nlm.nih.gov/pubmed/24352519.

Jorge Guerra, Jasim Uddin, Dawn Nilsen, James Mclnerney, Ammarah Fadoo, Isirame B Omofuma, Shatif Hughes, Sunil Agrawal, Peter Allen, and Heidi M Schambra. Capture, learning, and classification of upper extremity movement primitives in healthy controls and stroke patients. In *2017 International Conference on Rehabilitation Robotics (ICORR)*, pages 547–554. IEEE, 2017.

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

Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011.

Catherine E. Lang, Jillian R. Macdonald, and Christopher Gnip. Counting repetitions: An observational study of outpatient therapy for people with hemiparesis post-stroke. *Journal of Neurologic Physical Therapy*, 31(1):3–10, 2007. doi: 10.1097/01.npt.0000260568.31746.34.

B Ovbiagele, L B Goldstein, R T Higashida, V J Howard, S C Johnston, O A Khavjou, D T Lackland, J H Lichtman, S Mohl, R L Sacco, and et al. Forecasting the future of stroke in the united states: a policy statement from the american heart association and american stroke association., Aug 2013. URL https://www.ncbi.nlm.nih.gov/pubmed/23697546.

Zhiguang Wang, Weizhong Yan, and Tim Oates. Time series classification from scratch with deep neural networks: A strong baseline. In *2017 International joint conference on neural networks (IJCNN)*, pages 1578–1585. IEEE, 2017.