Applying Long-Short Term Memory Recurrent Neural Networks for Real-Time Stroke Recognition

Emanuele Ledda
Lucio Davide Spano
Manu_Ledda@hotmail.com
davide.spano@unica.it
Department of Mathematics and Computer Science, University of Cagliari
Cagliari, Italy

ABSTRACT

This note discusses how to build a real-time recognizer for stroke gestures based on Long Short Term Memory Recurrent Neural Networks. The recognizer provides both the gesture class prediction and the completion percentage estimation for each point in the stroke while the user is performing it. We considered the stroke vocabulary of the $1 and $N datasets, and we defined four different architectures. We trained them using synthetic data, and we assessed the recognition accuracy on the original $1 and $N datasets. The results show an accuracy comparable with state of the art approaches classifying the stroke when completed, and a good precision in the completion percentage estimation.

CCS CONCEPTS

• **Human-centered computing → Gestural input**; **User interface toolkits**; *Touch screens.*

KEYWORDS

stroke gestures, neural networks, real time recognition, long-short term memory

ACM Reference Format:
Emanuele Ledda and Lucio Davide Spano. 2021. Applying Long-Short Term Memory Recurrent Neural Networks for Real-Time Stroke Recognition. In *Companion of the 2021 ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '21 Companion), June 8–11, 2021, Virtual Event, Netherlands.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3459926.3464754

1 INTRODUCTION

The large availability of touch-sensitive screens in different interaction contexts fostered the adoption of stroke gestures for expressing commands in applications. In the literature, we can find different approaches for their recognition, employing a wide variety of techniques from Machine Learning [13, 16] to template-based recognition based on geometric techniques [1, 2, 10, 14, 17, 19, 20, 22]. The research for an accurate classification technique was limited to the correct gesture identification when the whole sample is available, increasing the accuracy level and decreasing the required time and space resources. However, providing the classification result at the end of stroke is not enough for supporting a proper user guidance system [4]. Indeed, to effectively guide a user, the interface requires information *while* s/he is performing the gesture, for representing the stroke portion the user has already completed (i.e., the *feedback*) and the possible completions for the current stroke the machine can correctly interpret (i.e., the *feedforward* [21]). In this scenario, we require a *real-time* a recognizer that, given the current (partial) input, identifies the most likely gesture label and the part of the stroke the user has to perform next. The latter information may be expressed in different ways, e.g., by labelling a set of discrete subparts (e.g., the sides of a triangle) or estimating the completion percentage in relation to the overall stroke shape. The support is supposed to update such information each time the interface receives a new point from the input device. Such a problem has gained interest in the latest research [3–5], but the accuracy results are significantly below the level reached considering the entire gesture classification and they put constraints on multi-stroke sequencing, e.g., not supporting parallel strokes.

In this note, we report on applying deep learning methods to the real-time recognition problem, returning the gesture label and the estimated completion percentage. We use a Recurrent Neural Network (RNN) architecture called Long-Short Term Memory (LSTM) using four different topologies for the classification task. We consider two well-known stroke gesture datasets: the $1 (including single-stroke gestures) and the $N (including multi-stroke gestures). The source code for the models and the data processing procedures are available in a GitHub repository [1]. We reach an accuracy in real-time recognition comparable to the state of the art approaches working on the entire stroke sample. The main drawbacks in using deep learning techniques are the need for a long training time, which may be relevant in a prototyping setting, and the data augmentation needed for building a robust recognizer, which requires an understanding of the classification process.

2 DATA REPRESENTATION AND PREPROCESSING

We consider two gestures vocabularies in our classification task. The first is the one provided by the $1-dataset, introduced in [22], which contains 330 repetitions of 16 single stroke gesture classes.

EICS '21 Companion, June 8–11, 2021, Virtual Event, Netherlands
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8449-0/21/06…$15.00
https://doi.org/10.1145/3459926.3464754

[1]https://github.com/unica-tesi-hci/deep-stroke

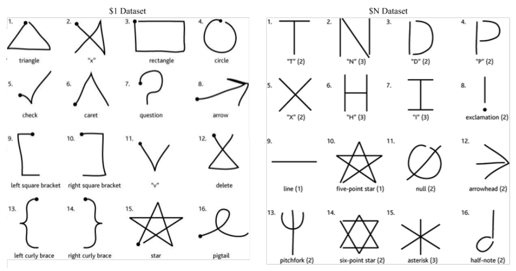

Figure 1: The gestures defined in the $1 [22] (left) and $N [1] (right) datasets.

The second is the $N-dataset, introduced in [1], which contains 600 repetitions of 16 single stroke gesture classes. Figure 1 shows the gestures available in both datasets.

Starting from the raw data, we modelled the strokes in both datasets as ordered sequences of 2D points, including a progressive stroke identifier (single stroke gestures have the same id for the whole gesture) and a boolean signalling the end of a stroke. We simulate the execution in real-time for all the stroke samples in both datasets for the classification task. The RNN model returns a gesture label and estimates the completion percentage for each point in the sample. We do not use the original $1 and $N datasets in the training phase, set but only as a test set. We discuss how we trained the model in Section 2.1.

The stroke samples vary in both scale and velocity. For this reason, it's common in all classification approaches to have a preprocessing step, including the sample recentering, rescaling in a 0 to 1 reference frame and resampling using a fixed number of points. Unfortunately, such steps require knowledge of the gesture's global properties, such as its length, total number of points, and bounding box. In a real-time recognition scenario, such global features are not known, so we cannot reduce the input variability through such standard preprocessing steps. Instead, we train the RNN to learn how to deal with the variability using data augmentation techniques and local resampling approaches.

2.1 Training Data

Many gesture datasets are available in different repositories (see for instance [11]), but only a few contain a number of samples suitable for training a deep network. Since acquiring such large datasets is often unfeasible for UI designers and engineers, we opted for using synthesised samples generated using the algorithm by Leiva et al. [9], which models the articulation of a stroke applying the Kinematic Theory of Rapid Human Movements [15]. Through such a technique, the generator can produce human-like stroke samples from the analysis of a few real ones. In Leiva's website[2] are available two synthetic datasets generated for the $1 and the $N vocabularies. We used them for the model training. More in detail, we split the synthetic datasets into two parts: we used 80% for training the models and the remaining part as the validation set for tuning the hyperparameters. Finally, we use as the test set the original $1 [22] and $N [1] datasets, which contain strokes performed by real users.

[2]https://luis.leiva.name/g3/

To obtain a model robust to the scale variability, we opted to augment the training set, including differently scaled versions of the same training instances. We generated five versions applying a random scaling assigned using a Gaussian distribution ($\mu = 0.2$, $\sigma = 0.045$). After that, we added a random translation for moving the sample into different screen portions, keeping the stroke entirely in the original bounding box. We estimated the parameters for the random scaling, translations and maximum bounding boxes during the hyperparameters tuning on the validation set. Finally, we applied a local resampling technique to make the training procedure robust to the variability in velocity, keeping a constant the Euclidean distance between two points of a stroke. We completed the dataset generation by adding a gesture label and the completion percentage for each stroke point in the dataset.

3 DEEP NETWORK ARCHITECTURE

The sequential nature of gesture data fits perfectly with Recurrent Neural Networks (RNN) [12], a particular type of model whose neurons' output is partially or totally connected to their input. Such a structure allows sequential data processing, providing output at each step and updating the hidden internal state for influencing further predictions at the next steps. We used the RNNs in a many-to-many configuration: at each step, the network outputs a probability distribution for each gesture in case of a classification task and a completion percentage in a regression task.

Among the available RNN architectures, we selected the Long Short-Term Memory (LSTM) [7], since it can trace both relevant information spotted in distant parts of the sequence (e.g., the beginning of the gesture), but it can also react to patterns in a small range of the current point. Such an architecture proved its effectiveness in tasks such as handwriting recognition [6].

To solve the real-time stroke recognition problem, we tested different simple network topologies. We need a model (either simple or composed) to solve two different tasks to be performed on both partial and complete gestures. The first is assigning a gesture label, i.e., a *classification*. The second is predicting the completion percentage (i.e., a *regression*) for distinguishing which part has been completed and which one is missing.

Figure 2 shows the network architectures. Two of them are composed of two separate models: one for the classification and one for the regression (see Figure 2, part A and B). The remaining two apply a multitask approach, where the same model provides both information (see Figure 2, part C and D)

3.1 Masking before the recurrent layer

To avoid global rescaling of the samples, we keep constant the distance between two points in a stroke (0.01 in a 0 to 1 reference frame). So, the sequences describing the strokes vary in length and the number of points, but the network training procedure needs fixed-size tensors for training. We used the maximum gesture length in the training set to estimate the maximum length of a gesture sequence, and padded each sequence with zero-tuples for constructing the correspondent tensor. Hence, we include a Mask Layer in all networks, which considers only the non-zero tuples valid gesture points.

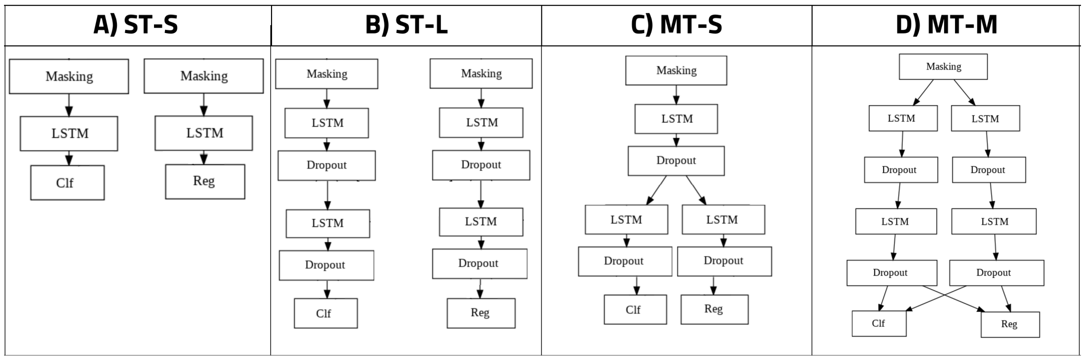

Figure 2: The four networks architectures used for real-time recognition. Two of them separate the gesture classification and the regression task, one using an LSTM component (part A, ST-S) while the other combines two LSTMs with dropout (part B, ST-L). The remaining models share part of the layers for performing both tasks, one splitting the output of an intermediate LSTM with dropout (part C, MT-S) and one adding two mutual connections (part D, MT-M)

3.2 Networks with task separation

The first architecture is a simple LSTM with 256 hidden neurons, followed by a fully connected layer. We call it Single Task - Simple (ST-S, see Figure 2, part A). The fully connected layer contains 16 neurons with a softmax activation for the classification task, providing the probability for the current sequence belonging to each gesture class. In the regression part, we use a single neuron with a sigmoid activation, which outputs a value between 0 and 1, indicating the predicted completion level.

The second topology we call Single Task - Long (ST-L, see Figure 2, part B) is similar to the first one, but it uses multiple LSTM layers. The first contains 256 hidden neurons, while the second one 64 neurons. Such configuration allows learning a more compact feature space for both the classification and the regression tasks. However, it leads to a higher overfitting probability. We included a dropout layer with a 0.2 rate after each LSTM layer for mitigating such a problem.

3.3 Multitask Networks

Multitasks Networks support the classification and the regression tasks using a single model. The first topology of this type is the Multitask - Simple (MT-S, see Figure 2, part C), which consists of an LSTM component of 256 hidden neurons followed by a dropout level (the same included in the two separate models in ST-L). After that, it has a branching with two dense layers, one used for the regression and one for the classification. This results in a shared topology, setting its weights for extracting two separate feature vectors for the different tasks.

The last topology we consider is called Multitask - Mutual (T-S, see Figure 2, part D). It is similar to the ST-L, but it contains two mutual connections between the dropouts levels after the last LSTM component. The topology includes two columns, each one extracting a different feature vector. Then, both extracted vectors feed the regression and the classifier layers.

3.4 Hyperpatameters

In our experiments, we used the cross-entropy loss [18] for training the classifier, the mean square error loss for the regression, and the Adam [8] optimizer, setting 10^{-4} as the learning rate and 128 as batch size. We needed to tune the momentum for getting a stable loss function while training since the default values were too high, leading to a large correction of the gradient at each step. We trained the model for 1000 epochs, using an NVidia GTX 1080 8Gb. Each epoch required between 2s to 4s for the ST-S architecture, 3.5s to 6s for the ST-L, 5s to 9s for MT-S and 7s to 12s for MT-M. This means that the time spent in training goes from roughly one hour for ST-S to three hours for MT-M.

4 RESULTS

In this section, we discuss the results we obtained in the real-time classification of the original $1 [22] and $N [1] datasets, containing strokes by real users, which we considered as our test sets. None of their samples has been used for training. We simulated a real-time performance of the strokes feeding the sample points to the network one by one as saved in the files.

We employed the following quality measures for assessing the recognition quality:

- **Accuracy** (number of correctly classified gesture points over the total points number) for evaluating the classification result. Higher percentages are better. We report three sub-versions of this measure, which consider the point correctly classified if the correct gesture label is the one having the highest probability (*best of 1*), among the top three (*best of 3*) or the top five (*best of 5*). Such measures are relevant since interfaces may show a set of possible completions for the current gesture (e.g., 3 or 5), and it is important to assess how many times the correct label is in that set.
- **Mean Squared Error** (the average squared difference between the estimated completion percentage and the actual value) for evaluating the completion estimation results. A low value of such an error means that the prediction is good (zero is perfect).

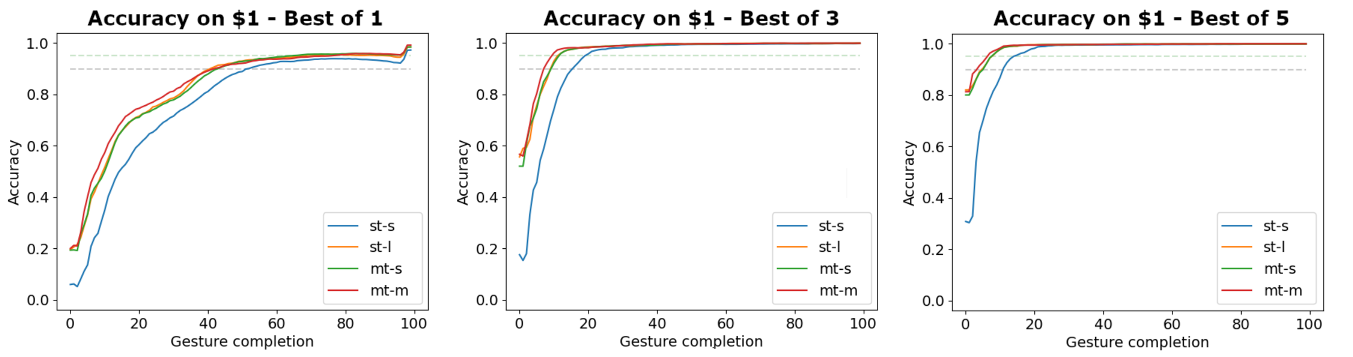

Figure 3: Real-time gesture classification accuracy for the $1 dataset [22]. We report the resulting accuracy for the best 1, 3 and 5 predicted labels. The dotted lines represent 90% and 95% accuracy.

4.1 Single Stroke Gestures

Figure 3 shows the classification results achieved by the four models we discussed in Section 3. The plots highlight that we achieved the worst performance with the simplest model (ST-S), which reaches an accuracy of 90.5% at 55% of the gesture completion. The ST-L and MT-S model perform better than ST-S, passing 90% of accuracy when the strokes are complete at 45%. At the beginning of the execution, the MT-M model performs slightly better than ST-L and MT-S, but it gets roughly the same performance after 45% of completion.

The final performance on the classification task is 94.52% for ST-S, 96.28% for ST-L, 96.76% for MT-S and 97.11% for MT-M. We outperform the result achieved in a real-time setting by Carcangiu and Spano [3], which was 88.2%. The performance is comparable with the results in the literature of the classification of the entire stroke. The $1 algorithm reached 97.1% [22] and the P$ 96.6% [19]. We recall that the offline classification is easier in general, given that we can decrease the input variability by recentering and rescaling it using a global bounding box, which is not available in a real-time scenario.

The results in the best of 3 and best of 5 classifications follow a trend similar to the best of 1 (see 3). The accuracy values show that the correct label is among the top-3 predictions already after recording the 10% of the stroke more than 95% of the samples, considering the ST-L, MT-S and MT-M models. Again, we have a lower performance for the ST-S model, which reaches 92% of accuracy about at the 20% of completion in the best of 3 case. This means that, even when the best candidate is wrong, the correct label is almost certainly inside the best three in all the models. So, from a UI engineering point of view, it's enough to manage three possible stroke completions between the beginning of the stroke and 45% of its completion and taking the best one after that for being sure at 90% to have correctly identified the stroke.

Figure 4 shows the Mean Squared Error (MSE) for the prediction of the completion percentage. The prediction error increases with the gesture completion, and we notice a steeper trend in the final part of the stroke (more than 60% of the completion) for the MT-M model, while ST-S produces higher errors between the 25% and the 42% of completion. ST-L and MT-S follow the best trend.

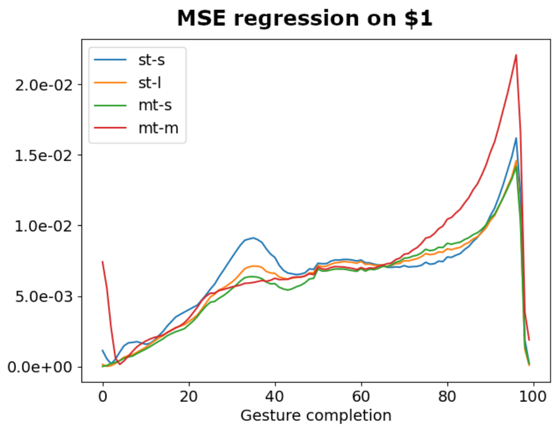

Figure 4: Mean Squared Error for the prediction of the stroke completion percentage for the $1 [22] dataset (the lower, the better)

Overall, the prediction is good for all the models. We have a mean error that stays between 7% and 14% in the MT-S model and between 7% and 12% for the other ones. The regression has a drift starting at 80% and finishing about at the end of the stroke, doubling the error value (in Figure 4 the error is squared). Such drift stops with a final drop, produced by the stroke completion flag, which clearly enforces a 100% prediction in the regression model. We get the best performance using the MT-S topology.

Unfortunately, we do not have many literature results for performance comparison on the gesture completion estimation. Other approaches discussing real-time recognition (e.g., [3–5]) predict semantic stroke sub-parts, such as the sides of a triangle or the arcs of a circle. However, given that for the $1 stroke set we have a maximum of 6 sub-parts, we can roughly estimate the maximum precision of these approaches in gesture completion percentage prediction at 16.6%, which is comparable with our regression results.

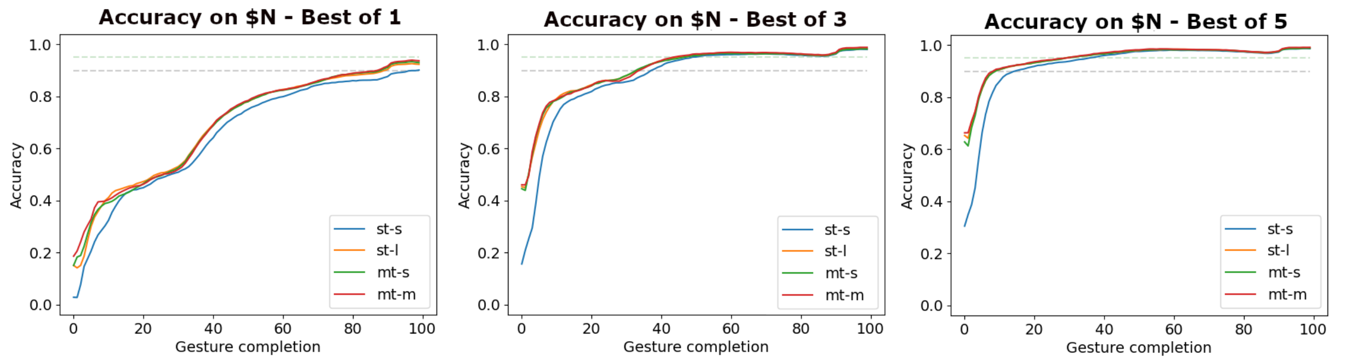

Figure 5: Real-time gesture classification accuracy for the $N dataset [1]. We report resulting accuracy for the best, the best 3 and best 5 predicted labels. The dotted lines represent 90% and 95% accuracy.

4.2 Multi Stroke Gestures

The model performance on the multistroke dataset is summarised in Figure 5. Overall, all models in such configuration reach a very high accuracy value quite late in the gesture performance. We get the 90% accuracy once the gesture is completed with ST-S, at 93% completion with ST-L, at 90% with MT-S and MT-M. However, a half gesture is sufficient for telling apart the 73% of strokes using ST-S, 76% using SL-L and MT-S and 77% using MT-M. The complete gesture recognition accuracy is 89.92% for ST-S, 92.44% for ST-L, 93.15% for MT-S and 93.75%. We are below the best values in state-of-the-art approaches using an offline setting, but the difference is about a couple of percentage points (e.g., 98% in [19], 96.4% in [1]).

Figure 6 shows the Mean Squared Error for the prediction of the completion percentage on the $N dataset. As happened for the single stroke gestures, the prediction error increases during the gesture performance. We register the drift and the drop in the final part of the gesture again, and both of them happen earlier: the drift starts about at 70%, while the drop starts at 85% of the gesture completion.

Overall, we still have a good prediction even if the error sets between 10% and 16% for the MT-M (which has the worst performance) and between 10% and 14% for the MT-S model (the best one).

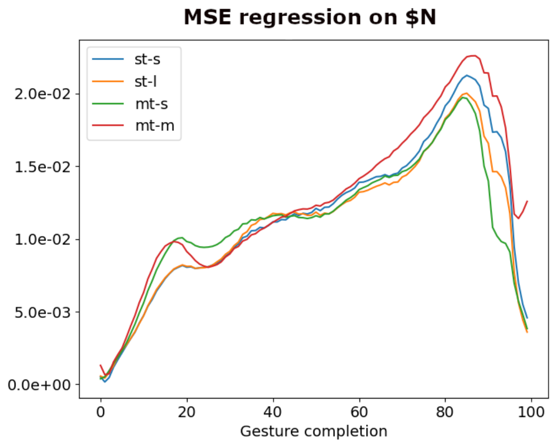

Figure 6: Mean Squared Error for the prediction of the stroke completion percentage for the $N [1] dataset (the lower, the better)

5 DISCUSSION AND CONCLUSION

In this note, we discussed how to build a recognizer for stroke gestures based on LSTM RNNs. We considered a real-time recognition scenario (also known as eager or online), where the recognizer has to provide a prediction for each point in the stroke while the user is performing it. We considered four simple network topologies, which are easy to set up using deep network libraries. In the considered a scenario, the user interface requires two pieces of information from the recognizer for building guidance systems based on feedback and feedforward: the predicted gesture class and an estimation of the gesture completion percentage.

To replicate the common situation where acquiring a dataset for training the network is not feasible for time or cost reasons, we used only synthetic samples for this task. Such a problem is one of the main barriers in applying deep learning approaches in gestural interfaces development.

The results discussed for both single and multi-stroke gestures show that even using simple topologies, we achieved a classification accuracy comparable with offline approaches and a good prediction of the completion percentage.

Our work's main limitation is that we did not consider a more complex architecture for the recognition network. We did it mainly because we are interested in understanding if LSTM models may be used as tools for this task, even for people that are not expert in deep learning. In particular, the regression may be improved by more complex network structures. There are also some drawbacks in using deep learning approaches we would like to mention here briefly. The first is training time. We needed an average time of 1 hour and a half for creating a recognizer. This is an important weakness for interface prototyping, given that the template-based approaches gained interest in the community since they allow building recognizes in no time. The second one is the data augmentation required for creating a robust recognizer. While the operations we performed on the original samples are quite trivial, they require a good understanding of the model's classification and regression process. If recognizers should be used as tools by UI designers and engineers, they may lack skills for building an appropriate dataset. The emphasis is shifted from the identification of recognition algorithms to the identification of the best training instances.

REFERENCES

[1] Lisa Anthony and Jacob O. Wobbrock. 2010. A Lightweight Multistroke Recognizer for User Interface Prototypes. In *Proceedings of Graphics Interface 2010* (Ottawa, Ontario, Canada) *(GI '10)*. Canadian Information Processing Society, CAN, 245–252.

[2] Lisa Anthony and Jacob O Wobbrock. 2012. $N-protractor: a fast and accurate multistroke recognizer. In *Proceedings of Graphics Interface 2012*. 117–120.

[3] Alessandro Carcangiu and Lucio Davide Spano. 2018. G-gene: A gene alignment method for online partial stroke gestures recognition. *Proceedings of the ACM on Human-Computer Interaction* 2, EICS (2018), 1–17.

[4] Alessandro Carcangiu, Lucio Davide Spano, Giorgio Fumera, and Fabio Roli. 2019. DEICTIC: A compositional and declarative gesture description based on hidden markov models. *International Journal of Human-Computer Studies* 122 (2019), 113–132.

[5] Stefano Dessì and Lucio Davide Spano. 2020. DG3: Exploiting Gesture Declarative Models for Sample Generation and Online Recognition. *Proc. ACM Hum.-Comput. Interact.* 4, EICS, Article 82 (June 2020), 21 pages. https://doi.org/10.1145/3397870

[6] Alex Graves and Jürgen Schmidhuber. 2009. Offline Handwriting Recognition with Multidimensional Recurrent Neural Networks. In *Advances in Neural Information Processing Systems*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou (Eds.), Vol. 21. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2008/file/66368270ffd51418ec58bd793f2d9b1b-Paper.pdf

[7] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural computation* 9, 8 (1997), 1735–1780.

[8] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[9] Luis A. Leiva, Daniel Martín-Albo, and Réjean Plamondon. 2015. Gestures à Go Go: Authoring Synthetic Human-Like Stroke Gestures Using the Kinematic Theory of Rapid Movements. *ACM Trans. Intell. Syst. Technol.* 7, 2, Article 15 (Nov. 2015), 29 pages. https://doi.org/10.1145/2799648

[10] Yang Li. 2010. Protractor: A Fast and Accurate Gesture Recognizer. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Atlanta, Georgia, USA) *(CHI '10)*. Association for Computing Machinery, New York, NY, USA, 2169–2172. https://doi.org/10.1145/1753326.1753654

[11] Nathan Magrofuoco, Paolo Roselli, Jean Vanderdonckt, Jorge Luis Pérez-Medina, and Radu-Daniel Vatavu. 2019. GestMan: a cloud-based tool for stroke-gesture datasets. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2019, Valencia, Spain, June 18-21, 2019*, José Ignacio Panach, Jean Vanderdonckt, and Oscar Pastor (Eds.). ACM, 7:1–7:6. https://doi.org/10.1145/3319499.3328227

[12] Larry R Medsker and LC Jain. 2001. Recurrent neural networks. *Design and Applications* 5 (2001).

[13] S. Mitra and T. Acharya. 2007. Gesture Recognition: A Survey. *IEEE Trans. Systems, Man, and Cybernetics, Part C* 37, 3 (2007), 311–324. https://doi.org/10.1109/TSMCC.2007.893280

[14] Corey Pittman, Eugene M. Taranta II, and Joseph J. LaViola. 2016. A $-Family Friendly Approach to Prototype Selection. In *Proceedings of the 21st International Conference on Intelligent User Interfaces* (Sonoma, California, USA) *(IUI '16)*. Association for Computing Machinery, New York, NY, USA, 370–374. https://doi.org/10.1145/2856767.2856808

[15] Réjean Plamondon. 1995. A kinematic theory of rapid human movements. *Biological Cybernetics* 72, 4 (01 Mar 1995), 295–307. https://doi.org/10.1007/BF00202785

[16] Siddharth S. Rautaray and Anupam Agrawal. 2015. Vision based hand gesture recognition for human computer interaction: a survey. *Artif. Intell. Rev.* 43, 1 (2015), 1–54. https://doi.org/10.1007/s10462-012-9356-9

[17] J. Reaver, T. F. Stahovich, and J. Herold. 2011. How to Make a Quick$: Using Hierarchical Clustering to Improve the Efficiency of the Dollar Recognizer. In *Proceedings of the Eighth Eurographics Symposium on Sketch-Based Interfaces and Modeling* (Vancouver, British Columbia, Canada) *(SBIM '11)*. Association for Computing Machinery, New York, NY, USA, 103–108. https://doi.org/10.1145/2021164.2021183

[18] Ambuj Tewari and Peter L Bartlett. 2007. On the Consistency of Multiclass Classification Methods. *Journal of Machine Learning Research* 8, 5 (2007).

[19] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2012. Gestures as Point Clouds: A $P Recognizer for User Interface Prototypes. In *Proceedings of the 14th ACM International Conference on Multimodal Interaction* (Santa Monica, California, USA) *(ICMI '12)*. Association for Computing Machinery, New York, NY, USA, 273–280. https://doi.org/10.1145/2388676.2388732

[20] Radu-Daniel Vatavu, Lisa Anthony, and Jacob O. Wobbrock. 2018. $Q: A Super-Quick, Articulation-Invariant Stroke-Gesture Recognizer for Low-Resource Devices. In *Proceedings of the 20th International Conference on Human-Computer Interaction with Mobile Devices and Services* (Barcelona, Spain) *(MobileHCI '18)*. Association for Computing Machinery, New York, NY, USA, Article 23, 12 pages. https://doi.org/10.1145/3229434.3229465

[21] Jo Vermeulen, Kris Luyten, Elise van den Hoven, and Karin Coninx. 2013. Crossing the bridge over Norman's Gulf of Execution: revealing feedforward's true identity. In *Proceedings of CHI 2013*. ACM, 1931–1940.

[22] Jacob O. Wobbrock, Andrew D. Wilson, and Yang Li. 2007. Gestures Without Libraries, Toolkits or Training: A $1 Recognizer for User Interface Prototypes. In *Proceedings of UIST 2007 (UIST '07)*. ACM, New York, NY, USA, 159–168. https://doi.org/10.1145/1294211.1294238