LONG SHORT TERM MEMORY NEURAL NETWORK FOR KEYBOARD GESTURE DECODING

Ouais Alsharif, Tom Ouyang, Françoise Beaufays, Shumin Zhai, Thomas Breuel, Johan Schalkwyk

Google

ABSTRACT

Gesture typing is an efficient input method for phones and tablets using continuous traces created by a pointed object (e.g., finger or stylus). Translating such continuous gestures into textual input is a challenging task as gesture inputs exhibit many features found in speech and handwriting such as high variability, co-articulation and elision. In this work, we address these challenges with a hybrid approach, combining a variant of recurrent networks, namely Long Short Term Memories [1] with conventional Finite State Transducer decoding [2]. Results using our approach show considerable improvement relative to a baseline shape-matching-based system, amounting to 4% and 22% absolute improvement respectively for small and large lexicon decoding on real datasets and 2% on a synthetic large scale dataset.

***Index Terms*—** Long-short term memory, LSTM, gesture typing, keyboard

1. INTRODUCTION

The Word-Gesture Keyboard (WGK) [3], first published in early 2000s [4, 5] has become a popular text input method on touchscreen mobile devices. In the last few years this new keyboard input paradigm has appeared in many products including Google's Android keyboard. With a WGK, rather than tapping letter by letter, the user slides a finger through letter keys on a touchscreen keyboard to enter text at a word level. For example, to write the word *the* the user may, approximately, land a finger on the T key, slide to the H key, continue to the E key, and then lift up. The single-stroke gesture makes a spatial representation of the word *the* to be recognized by the keyboard decoder. Figure 1 depicts such interaction.

Word gestures produced with gesture typing are inherently ambiguous. As to reach a particular letter on the keyboard, a gesture stroke inevitably runs across other letters that are not part of the intended word. Also, on the other end, a gesture may not touch letters that are part of the intended word. In addition to ambiguity, gesture typing presents coarticulation-like effects, as the way a character is touched depends on its surrounding characters. Aside from these difficulties, gesture typing presents several challenges in terms of decoder efficiency, accuracy and robustness. Efficiency is important because such solutions tend to run on mobile devices which permit a smaller footprint in terms of computation and memory. Accuracy and robustness are also crucial, as users are not expected to gesture type words with high accuracy. Since most users would be using a mobile device for input, it is highly likely that gestures would be offset, noisy or even mixed with conventional typing.

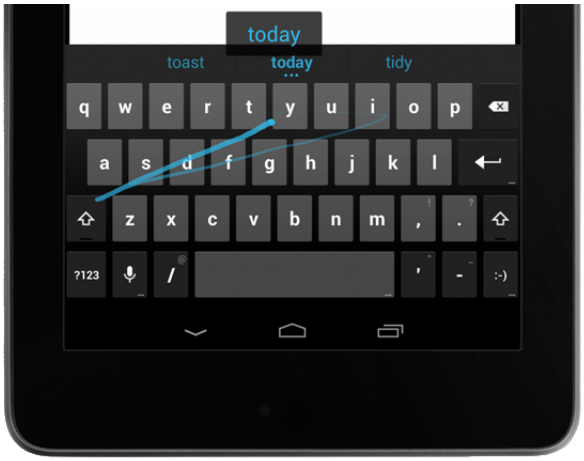

Fig. 1. An example keyboard with the word *Today* gestured.

Conventional methods for gesture typing such as shape matching [4] tend to be highly accurate on clean data, however they suffer heavily when the input gestures are noisy, or when they are segmented into multiple gestures. On the other hand, machine learning models, particularly recurrent neural nets, have shown large gains in recognizing noisy, ill-segmented inputs, such as hand written text [6] and speech [7].

Recurrent Neural Networks [8] are set apart from conventional feed-forward neural networks by allowing cycles in the computational graph represented by the network. These cycles allow the network to have dynamic contextual windows, as opposed to fixed size windows, used by standard feed forward models when addressing sequential problems. This dynamic window contributes to improved accuracy of such models on challenging tasks, such as large vocabulary speech recognition [7].

Despite their potential for incorporating long-term contexts, vanilla RNNs trained with conventional Back Propagation Through Time (BPTT) perform poorly on sequence modelling tasks. This is primarily due to the gradient vanishing and explosion problems [9]. Long Short Term Memories (LSTMs) [1] on the other hand address these issues through a memory block containing a memory cell which has a self connection storing the temporal state of the network, allowing a steady gradient during training. Figure 2 depicts the architecture of an LSTM memory block.

In this paper, we exploit the conceptual similarity between keyboard gesture decoding and speech recognition, by concretely showing how a variant of LSTMs [7] can be used to tackle the gesture typing problem. We find that relatively simple, shallow networks present superior performance relative to an improved variant of a state of the art Google Android keyboard commercial product. In addition to their superior performance, on clean data, we show how LSTMs are able to robustly recognize "noisy" data that the baseline recognizer is unable to capture.

2. PROBLEM DEFINITION

Here, we formally define the gesture typing problem. A keyboard gesture decoder seeks to learn a function $f : \mathbb{R}^{d \times T} \rightarrow \mathcal{W}$[1] where T is the length of the gesture, d is the number of features in the each input frame and $\mathcal{W}$ is the set of permitted words. Typically, the d features associated with an input x contain signals like: the keys the finger touches, current time from last gesture and type of gesture (a down press or an up lift of a finger).

3. RELATED WORK

Variants of recurrent neural networks have been applied to various sequence prediction and labelling problems such as speech recognition [7], handwriting recognition [6], grammatical inference [10], among others. Arguably, the most significant success recurrent neural nets (LSTMs in particular) had so far was for automatic speech recognition [11], where they surpass conventional feed-forward neural networks for acoustic modelling tasks. It is notable that, while LSTMs do outperform conventional neural networks, vanilla RNNs do not [7], with bidirectional models [6], (models which operate on the input signal and its reverse simultaneously) outperforming unidirectional models.

While effective for sequence classification tasks, in order to use LSTMs to map input sequences to shorter output sequences (e.g., mapping acoustic frames to phoneme or grapheme sequences), one needs an alternative loss function to conventional frame-wise cross-entropy loss. Concretely,

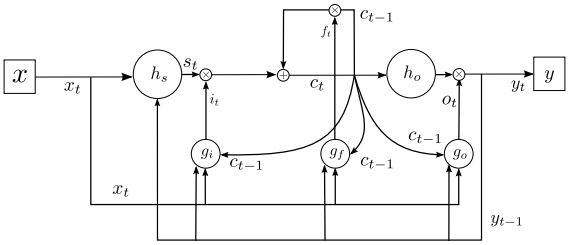

Fig. 2. The architecture of a single memory block of an LSTM.

the Connectionist Temporal Classification (CTC) loss function [12] can be used for this purpose. This function attempts to increase the probability of all frame wise transcriptions that result in the target output sequence after removing blanks and repetitions. Results using CTC for speech recognition have been mixed. While the state of the art result on TIMIT [11] is achieved with a CTC type-loss, recent attempts for end-to-end training on the wall-street journal with a similar approach resulted in comparable Word Error Rates (WER) to a conventional feed forward neural net baseline [13].

In a different strand of research, methods for keyboard gesture recognition, and more generally, gesture recognition have been largely reliant on geometric distance techniques [4, 5], HMMs [14] or Dynamic Time Warping [15], among others. For a comprehensive survey on gesture recognition, consult [16].

4. METHODOLOGY

In this section, we describe our particular approach in more detail.

4.1. Long Short Term Memory

To address the gesture typing problem, we use an LSTM trained with a CTC loss function. Unlike the work in [7] we do not use recurrent layers or projection layers in our particular architecture. For an input sample $x \in \mathbb{R}^{d \times T}$, an LSTM computes the following function[2]:

$$s_t = h_s(W_s(y_{t-1} + x_t)) \quad (1)$$
$$i_t = g_i(W_i(y_{t-1} + x_t + c_{t-1})) \quad (2)$$
$$f_t = g_f(W_f(y_{t-1} + x_t + c_{t-1})) \quad (3)$$
$$c_t = i_t \odot s_t + c_{t-1} \odot f_t \quad (4)$$
$$o_t = g_o(W_o(x_t + y_{t-1} + c_t)) \quad (5)$$
$$y_t = o_t \odot h_o(c_t) \quad (6)$$

[1] Note that T is different for different inputs.

[2] We omit biases for clarity of presentation.

where $x_t \in \mathbb{R}^d$ is the t-th column of the input matrix x. W_s, W_i, W_f, W_o are the cell's parameters. i_t, f_t, o_t are respectively the input, forget and output gates' outputs. g_i, g_f and g_o are the input, forget and output gates activation functions. h_s and h_o are the input and output activations and $\odot$ is the element wise product. After passing the entire sequence through this function, the LSTM produces an output $y \in \mathbb{R}^{|C| \times T}$ where C is the set of permitted characters.

4.2. Connectionist Temporal Classification Loss

The CTC loss function can be used to train recurrent networks on unaligned targets. This is accomplished through maximizing the sum of probabilities of all frame-wise sequences that correspond to the target sequence. Concretely, this loss is given as follows:

$$\mathcal{L}(x, z) = -\log(p(z|x)) \tag{7}$$

$$p(z|x) = \sum_{\pi \in \mathcal{A}(z)} p(\pi|x) \tag{8}$$

$$p(\pi|x) = \prod_{t=1}^{T} y_{\pi_t}^t \tag{9}$$

where x is the input sequence, z is the target transcription, $\mathcal{A}(z)$ is the set of all CTC transcriptions of a target transcript (e.g., for the word *data*, the transcriptions may include *daata*, *datta*, *dddata*, etc). y is the output of the LSTM or more generally, a recurrent network.

To compute the above loss, we make use of the forward-backward algorithm [17], where we find:

$$p(z|x) = \sum_{s=1}^{|\hat{z}|} \frac{\alpha_T(s)\beta_T(s)}{y_{\hat{z}_s}^T} \tag{10}$$

where α and β are respectively the forward and backward variables, defined as in [12] and can be computed via dynamic programming. $\hat{z}$ is the same as sequence z with blanks inserted in the beginning, end and between symbols. z_s is the sequence containing the first s frames of z.

Using the CTC loss function during training allows us to train the network to output characters directly, without the need for HMM states.

4.3. Finite State Transducers

After training, an LSTM network produces a matrix $y \in \mathbb{R}^{|C| \times T}$. In order to constrain the decoded result to a set of limited words $\mathcal{W}$, we construct a trie-shaped lexicon FST L, similar to the one in Figure 3. We compose this FST with another c FST that maps CTC blanks into FST epsilons. We decode the resulting matrix using the composed $c \circ L$ FST with a standard beam search [18], where arc transitions costs are the context-independent log probabilities from the matrix y. Note that while we do not use a language model in this particular work, adding one is straightforward, through a simple composition with a G FST containing the language model information.

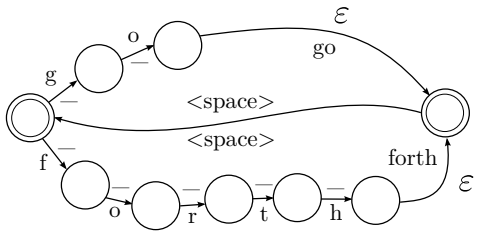

Fig. 3. An example lexicon FST for the words *go* and *forth*. The _ symbol stands for emitting nothing, ε is the FST epsilon.

5. EXPERIMENTS

We evaluate our methodology on three different datasets. Two of which were collected from real users, and one synthetic. We compare our method to a strong baseline, namely an advanced version of the technology currently in use for Android KitKat Keyboard gesture recognition.

5.1. Data

We evaluate the performance of our method on three datasets. The first (called *Salt*) is a relatively small dataset, containing 14,500 words in total, with 120 unique words, collected from 40 opt-in individuals in a wizard of oz fashion, modelled after the Pepper study [19]. The second dataset (called *ALK*) contains 50,000 words with 5,450 unique words. The words in this dataset were anonymized real gestures collected from opt-in Google employees. Due to heavy preprocessing, this dataset contained roughly 70% words that had a single down and up taps and 30% which had multiples. This is particularly of interest, as we find the baseline system to be unable to capture the multi-tap gestures. The third dataset was synthetically generated from the Enron dataset [20]. We preprocessed the data by narrowing down to 89 users that had a reasonable amount of sent mail, heuristically removing replied-to and forwarded text, removing URLs and then discarding messages that had more than 1000 words. Then, from the resulting data, we generated synthetic gestures, by connecting the characters within a word using an algorithm that minimizes jerk, which closely fits human motor control. We also allowed for variability in the length of the sequences. After this preprocessing, the final dataset contains ∼138,000 words with 8,256 unique words. For all datasets, we lower-cased all words and removed words with characters other than a-z.

We split all three datasets into training/testing sets. With the *Salt* dataset split into 13,000 training points for training and 1500 for testing. For the *ALK* dataset, we split into 45,000 for training and 5,000 for testing. As for the *Enron* dataset, we split into 124,000 points for training and 14,000 for testing. These statistics are summarized in Table 1.

Dataset	Salt	Alk	Enron
Data Type	Real	Real	Synthetic
Unique Words	120	5,450	8,256
Training set size	13,000	45,000	124,000
Testing set size	1,500	5,000	14,000

Table 1. Dataset Statistics

The input to our gesture keyboard app contains an x, y position, time since last gesture and gesture type (move, up, down). We augment these inputs with a 30 dimensional boolean vector in which index $i = 1$ for key i if the x, y position fell within key boundaries. The 30 buttons include 26 characters for the English alphabet in addition to commas, apostrophes, dots and space. These 34 features form the frame-wise inputs to our LSTMs.

We preprocess all datasets using frame-wise global contrast normalization (i.e., removing the mean and dividing by standard deviation of the frame vectors).

5.2. Model

We experimented with various LSTM architectures, with varying depth, width and directionality. In all experiments, we let the set C be equal to be the set of small case English characters in addition to the apostrophe, comma and dot characters, except for the *Salt* experiment, where we set $C = \mathcal{W}$ directly. We initialize all weights in our networks to be Gaussian distributed with a mean of 0 and a standard deviation of 0.01. The LSTMs used a hyperbolic tangent activation unit for cell inputs and outputs and logistic units for all the gates. As is conventional with LSTMs, we use gradient clipping to avoid the gradient explosion problem. All networks were trained with asynchronous SGD on a parallel computing cluster [21]. We train the neural network with a learning rate set to 0.01. For our L FST, we use a lexicon created from 100,000 words selected from an independent lexicon. We use this FST for the *ALK* and *Enron* experiments. As for the *Salt* experiments, we constrain the lexicon to the 120 words in the dataset.

5.3. Results

For each of the datasets, we report word recognition accuracy. For all LSTM experiments, we report model sizes, as these tend to be important factors for applications on embedded devices. We found deeper models, with two LSTM layers to perform quite comparably to shallow ones, so we omit them from results. Moreover, we find Bidirectional LSTMs (BLSTMs) to consistently outperform Unidirectional LSTMs (ULSTMs). We also found we were unable to train ULSTMs on the *Salt* dataset. We believe this is due to the small number of unique words in that particular dataset, as the LSTM does not have sufficient data to generalize. The *ALK* and *Enron* experiments showcase that point, as ULSTMs perform increasingly better with larger amounts of training data. We also consistently find learning rate decay to be useful. Particularly, for our most accurate BLSTM model, learning rate decay led to nearly 5% absolute improvement in WER on *ALK*. Table 2 presents the results of these experiments.

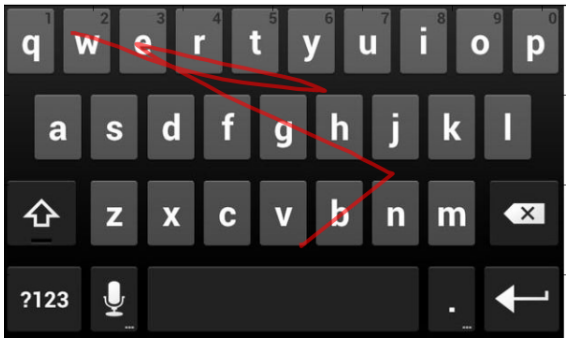

Fig. 4. An example real gesture of the word *when* which our LSTM recognizes correctly while the baseline system fails to. Notice the co-articulation-like effect after the character n which confuses the baseline system.

Model	Salt	Alk	Enron	size
ULSTM-34-400*	-	76.0%	92.5%	1.5m
ULSTM-34-400	-	76.3%	90.2%	1.5m
ULSTM-34-200	-	77.8%	90.0%	451k
ULSTM-34-100	-	76.6%	89.6%	148k
ULSTM-34-50	-	70.7%	85.9%	55k
BLSTM-34-400*	**92%**	**89.2%**	**93.5%**	1.5m
BLSTM-34-400	89%	84.2%	92.3%	1.5m
BLSTM-34-200	88%	83.6%	91.7%	451k
BLSTM-34-100	86%	82.8%	91.1%	148k
BLSTM-34-50	85%	75.7%	88.2%	55k
Baseline	88%	67%	91.2%	N/A

Table 2. Recognition Accuracy and model sizes for different LSTM models on the Salt, Alk and Enron datasets. * indicates results after learning rate decay.

Arguably the most interesting aspect of our results, is that we find the baseline system to perform well on clean data ($\sim$ 92%) on *ALK*, but quite poorly ($\sim$ 0%) on noisy data (in which the user multi-tapped , mixed gestures or didn't raise their finger at the right time). Where LSTM models perform comparably on both noisy and clean data. Figure 4 depicts an example noisy gesture which our LSTM recognizes while the baseline system fails to.

6. CONCLUSIONS

To the best of our knowledge, we presented in this paper the first successful application of a recurrent network based architecture to address the keyboard gesture decoding problem. We have shown the possibility of training these networks on unsegmented data, where we utilized a CTC type loss function during training. Furthermore, we showed that in combination with Finite State Transducers, these methods can scale to lexicons with tens of thousands of words.

7. REFERENCES

[1] Sepp Hochreiter and Jürgen Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[2] Mehryar Mohri, "Finite-state transducers in language and speech processing," *Computational linguistics*, vol. 23, no. 2, pp. 269–311, 1997.

[3] Shumin Zhai and Per Ola Kristensson, "The word-gesture keyboard: reimagining keyboard interaction," *Communications of the ACM*, vol. 55, no. 9, pp. 91–101, 2012.

[4] Shumin Zhai and Per-Ola Kristensson, "Shorthand writing on stylus keyboard," in *SIGCHI*. ACM, 2003, pp. 97–104.

[5] Per-Ola Kristensson and Shumin Zhai, "Shark 2: a large vocabulary shorthand writing system for pen-based computers," in *Proceedings of the 17th annual ACM symposium on User interface software and technology*. ACM, 2004, pp. 43–52.

[6] Marcus Liwicki, Alex Graves, Horst Bunke, and Jürgen Schmidhuber, "A novel approach to on-line handwriting recognition based on bidirectional long short-term memory networks," in *ICDAR*, 2007.

[7] Haşim Sak, Andrew Senior, and Françoise Beaufays, "Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition," *arXiv preprint arXiv:1402.1128*, 2014.

[8] Ronald J Williams and David Zipser, "A learning algorithm for continually running fully recurrent neural networks," *Neural computation*, vol. 1, no. 2, pp. 270–280, 1989.

[9] Yoshua Bengio, Patrice Simard, and Paolo Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Transactions on Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[10] Steve Lawrence, C Lee Giles, and Sandiway Fong, "Natural language grammatical inference with recurrent neural networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 12, no. 1, pp. 126–140, 2000.

[11] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton, "Speech recognition with deep recurrent neural networks," in *ICASSP*. IEEE, 2013, pp. 6645–6649.

[12] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber, "Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks," in *ICML*. ACM, 2006, pp. 369–376.

[13] Alex Graves and Navdeep Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *ICML*, 2014, pp. 1764–1772.

[14] Andrew D Wilson and Aaron F Bobick, "Hidden markov models for modeling and recognizing gesture under variation," *PAMI*, vol. 15, no. 01, pp. 123–160, 2001.

[15] Miguel Angel Bautista, Antonio Hernández-Vela, Victor Ponce, Xavier Perez-Sala, Xavier Baró, Oriol Pujol, Cecilio Angulo, and Sergio Escalera, "Probability-based dynamic time warping for gesture recognition on rgb-d data," in *Advances in Depth Image Analysis and Applications*, pp. 126–135. Springer, 2013.

[16] Sushmita Mitra and Tinku Acharya, "Gesture recognition: A survey," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 37, no. 3, pp. 311–324, 2007.

[17] Lawrence Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[18] Stuart Russell and Peter Norvig, "A modern approach," 1995.

[19] Shiri Azenkot and Shumin Zhai, "Touch behavior with different postures on soft smartphone keyboards," in *Proceedings of the 14th international conference on Human-computer interaction with mobile devices and services*. ACM, 2012, pp. 251–260.

[20] Bryan Klimt and Yiming Yang, "Introducing the enron corpus," .

[21] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al., "Large scale distributed deep networks," in *NIPS*, 2012, pp. 1223–1231.