An Intelligent Keyboard Framework for Improving Disabled People Computer Accessibility

Karim Ouazzane, Jun Li, and Hassan B. Kazemian

T10-03, Tower Building, London Metropolitan University,
166-220 Holloway Road, London, UK N7 8DB
{k.ouazzane,jul029,h.kazemian}@londonmet.ac.uk

Abstract. Computer text entry may be full of noises – for example, computer keyboard users inevitably make typing mistakes and their typing stream implies all users' self rectification actions. These may produce a great negative influence on the accessibility and usability of applications. This research develops an original Intelligent Keyboard hybrid framework, which can be used to analyze users' typing stream, and accordingly correct typing mistakes and predict users typing intention. An extendable Focused Time-Delay Neural Network (FTDNN) n-gram prediction algorithm is developed to learn from the users' typing history and produce text entry prediction and correction based on historical typing data. The results show that FTDNN is an efficient tool to model typing stream. Also, the computer simulation results demonstrate that the proposed framework performs better than using the conventional keyboard.

Keywords: Focused Time-Delay Neural Network, Intelligent Keyboard Hybrid Framework.

1 Introduction

It is inevitable that users will make typing mistakes, which is particularly the case for disabled users. These are different kinds of mistakes such as spelling errors and adjacent key press errors etc. [1] [2]. A series of research based on words vocabulary which apply both, neural network and language modeling have been carried out; Bengio and Ducharme [3] suggested a model using neural network probabilistic language modeling to learn distributed representation for words that allow each training sentence to inform the model about exponential number of semantically neighboring sentences. Schwenk and Gauvain [4] addressed a related problem further by carrying out probability estimation in a continuous space and enabling a smooth interpolation of the probabilities. However, due to the curse of dimensionality in the discrete space representation, they still have to narrow the vocabulary by using a shortlist which damages the prediction accuracy and fail to learn a long-span language model with n >> 3 gram. An alternative way is to install filters which modify the control signals generated by the device. Such filters can have a significant effect on the speed and accuracy with which a device can be used. Attempts have also been made by IBM to devise intelligent mechanisms that could adjust the settings of the

L. Iliadis and C. Jayne (Eds.): EANN/AIAI 2011, Part I, IFIP AICT 363, pp. 382–391, 2011.

keyboard accessibility features by detecting the usage problems as introduced in Trewin and Pain [5].

Dasher [6] is an information-efficient text-entry interface, driven by natural continuous pointing gestures [7]. It is based on language model prediction, through which the space of interface is determined to each piece of text. It is useful to the users who operate a computer onehanded, by joystick, touch screen, trackball or mouse, which might be an inspiration for QWERTY keyboard [8] tools development. Although the Dasher project is a good step forward, it still has some limitations. For example, Dyslexia can cause significant problems in remembering even short sequences of numbers in the correct order. Those types of disability frequently cause typing mistakes, which haven't been well solved. ProtoType [9] is a piece of software used to type text into other programs such as a word processor based o lists of words, which includes word prediction, spelling correction and word banks. ProtoType is designed to improve spelling for people with dyslexia or spelling difficulties, but it is dysfunctional to correct the keystrokes mistakes made by most motor disabled people. Research works have been carried out to address this problem, these include Metaphone [10] and n-grams [11], each of them may have its unique features, but they all have limitations.

Intelligent models such as neural network models have been implemented in various directions, however, they are hardly seen to apply to noisy text entry processing such as user typing stream. Moreover, although efforts have been made in multiple directions such as language modeling, natural language processing and user interface design, those technologies, if used alone, will fail to meet the user's particular needs which have been presented above. It is also worth arguing that combination of models such as Jianhua & Graeme word prediction model [12] emphasizes excessively on providing a global method, and lack 'user-oriented' feature. Furthermore current models are short of self-adaptive ability (i.e. learning ability), and fail to fully recognize the right patterns from user's distinct performance.

Ouazzane et al. [1] presented a hybrid framework based on machine learning model to offer an efficient solution for people having difficulties using QWERTY keyboard; it integrates neural network, language model and natural language processing technologies, to provide users with two fundamental functions, namely, word prediction and typing correction. Also, Li et al. [13] combined distinct word correction algorithms, using neural network approach, to produce an optimal prediction. It was demonstrated that neural network, as a learning means, can provide an optimum solution through combining distinct algorithms in order to achieve a high ranking performance. Hence, the main purpose of this research is to develop an Intelligent Keyboard hybrid novel framework which is based on the combination of multiple technologies (i.e. neural network and language modeling) and therefore to put all merits of those distinct solutions. It is desirable to develop a solution that is evolutionary and adjustable, which can learn from users' past mistakes and then to predict and/or correct these mistakes. In order to achieve the text correction and prediction, in this investigation a Focused Time-Delay Neural Network (FTDNN) [14] is chosen, which can represent the unclear and complex relationship between current typed sequence and its precedent one. In this paper, the Intelligent Keyboard hybrid system's case study is developed and the conclusion of the work alongside future recommendations is provided at the end.

2 Intelligent Keyboard Hybrid Framework

As to cognitive tasks it is shown that rather than seek solutions based on the symbolic Artificial Intelligence, a more potentially useful approach is to build a structured connectionist model or a hybrid system. Then it is able to combine more functions for some specific purposes, which includes four fundamental elements, namely, Environment Analysis, Learning Elements, Knowledge Base and Performance element [15]. All of these units could be divided into more subdivisions to form a highly efficient hybrid system. Therefore, an Intelligent Keyboard (IK) hybrid system which combines neural network, statistics and natural language model together, is designed and intends to provide users with fundamental functions such as typing prediction and correction. User's typing data stream can be checked, rectified, and predicted in sequence by going through each unit following user's typing process. Through this way, the typing stream's noises are filtered significantly and the language interaction between a user and a computer becomes smoother. Multiple units and a database separated by a long-term memory and a short-term memory have been presented according to the technologies. The architecture is shown in Figure 1, which includes four processing units: Text prediction, Inference, Natural Language Processing and Error correction units; and two additional modules: User interface module and Noise process module to enable the interaction of the user with the outside environment (i.e., computer keyboard).

The two additional modules function as data pre-processing, post-processing and interaction interface. They correspond to machine learning model's Environment Element and part of Performance Element respectively. The Knowledge Base element is represented by Long-term Memory and Short-term Memory. The rules inferred from Inference Engine and some other facts such as user profile and frequently used texts, are saved in the Long-term Memory. Other facts such as recently used new words are stored in the Short-term Memory which will be transferred to the long-term memory if a certain threshold is reached. The framework is invoked by user's key strokes. As much of data stream typing could be un-preprocessed, incomplete and noisy, for example, a long key press generates more than one Window's message, the data stream needs to undergo Noises Processing module first. Through this module the input vectors are further exploited, which would include the key-up signals, key-down signals, the time difference between two consecutive strokes and so on. The definition of noises can be given according to the user profile. Subsequently, a representation vector which includes time stamp and Virtual Key Code (VKC) [16] message is chosen to be sent to the processing units, namely, Text Prediction unit and Error Correction unit. Both units process the vectors based on the association rules, dictionaries and some other facts retrieved from the memory.

Text prediction unit is composed of different algorithms developed based on different scientific methods such as statistics and phonemics while Error Correction unit is designed based on users' performance. Firstly, a spell checker function is used to detect if a mistake had occurred. In the case of no mistakes being traced, the unit processing is stopped and the result is passed on to Inference Engine. Otherwise, the function such as motor checker, to process motor disability errors, would be evoked if spell checker fails to present a result. These two units (i.e. Text Prediction unit and Error Correction unit) can process data stream simultaneously. The typing mistakes

which are still under doubt are further checked by Natural Language Processing unit to check syntax and semantics' errors. Finally, the results are refurbished and shown to the user by User Interface unit.

The results (e.g. a list of words) generated from Text Prediction unit and Error Correction unit, which are usually more than one, are presented to Inference Engine unit. The Inference Engine unit ranks the results based on their probabilities to generate a word-list or directly presents a highest ranking presentation to the user. The user's feedback such as selections and correction actions is recorded by the inference algorithms (here is a neural network algorithm), and transferred to rules or rewards to be stored into the memory.

Shown as in Figure 1, an input of a sentence is a process passing through different structure status from letter, word to sentence, during which distinct units are evoked up according to the structure status' changes.

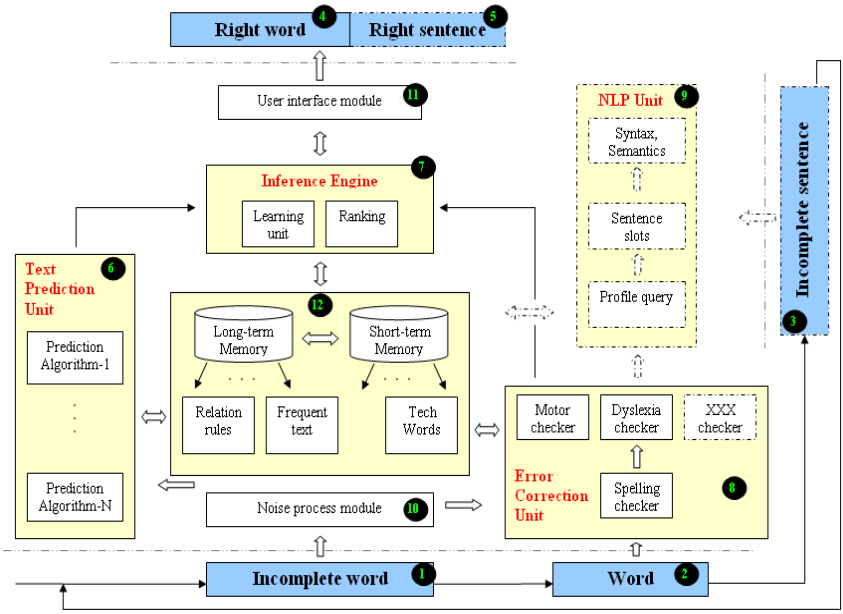

Fig. 1. The Architecture of Intelligent Keyboard; blue boxes and their connections represent the system's input and output process. The processing units from left to right, which has been marked as light yellow (or light grey in the case of black/white printing), are named as Text prediction unit (No. 6), Inference engine unit (No. 7), Error correction unit (No. 8), and Natural Language Processing (NLP) unit (No. 9). Data storage (No. 12) is divided into short-term memory and long-term memory, where the temporary and permanent information are stored. There are two additional modules: Noise process module (No. 10) and User interface module (No. 11), which are responsible for the interaction with the outer environment such as keyboard.

3 Intelligent Keyboard Framework Demonstration Using N-Gram Focused Time-Delay Neural Network Modeling

3.1 Modeling Data Sets

As illustrated above, IK framework provides two fundamental functions namely, prediction and correction. Typing stream prediction and correction can be achieved based on historical data by using neural networks through designing two models; or can be developed using a single neural network architecture if the correction can be considered as one specific case of prediction – the model produces the right symbol based on the inaccurate historical data. At this point a model based on Focused Time-Delay neural network modeling associated with different datasets is designed and implemented in the following sections. The used datasets are shown below;

♦ **Dataset one:** a novel – 'Far from the Madding Crowd' was written by Thomas Hardy [17]. It has been used as a testing sample by some compression algorithms such as PPM* [18]. The version used here is extracted from Calgary Corpus [19].

♦ **Dataset two:** it is extracted from a charity keystroke log. From the reflected keystroke log, the typing mistakes are predominantly about adjacent key press and prolong key press errors. The keystroke recording tool used in this research is KeyCapture software [20], which has been modified and adjusted for the purpose of this research. It runs in background under Windows environment to collect keystrokes without interfering with user's work.

3.2 N-Gram Focused Time-Delay Neural Network Modeling

Studying user's typing behavior would require the network to study user's history. FTDNN is suitable for time-series prediction. However, a comprehensive research on Focused Time-Delay Neural Network language modeling has never occurred. In the following sections an extendable FTDNN n-gram prediction is developed to predict noise-free and typing stream datasets.

♦ **FTDNN n-gram prediction definition:** let's assume existing string $S = \{s_1.s_i.s_j.s_k.s_m \mid i \le j \le k \le m\}$ and $(j-i) = n, (k-j) = l$, where s_i, s_j, s_k, s_m are symbols and *i, j, k, l, m, n* are natural numbers, if one builds a relation $R_n = \{x, y \mid x = (s_i ... s_j)_n \rightarrow y = (s_{j+1} -> s_k)_l\}$, then the relation is defined as *n*-gram's $l-$prediction; if one considers the special case $l = 1$, then the relation is called *n*-gram's one-prediction, or *n*-gram prediction for short. For example, given string *S*= *{student}*, some *2*-gram prediction cases are,

'st' → 'u'
'tu' → 'd' …

♦ **Symbol-distribution definition:** Given a certain ranking level *m* and a symbol set $A = \{a, ..., z, space\}$, one defines the n-gram Symbol-Distribution in ranking level *m* as $D_n^m = \{x, y \mid x_i -> y_i\}$, where $x_i \in$ symbol set *A*, and y_i is the level *m* Hitting Rate corresponding to each symbol.

Due to the system environment's limitations in this research, rather than adopting the whole dataset, a chunk of data ranging from zero to 100k is selected from the dataset one. The dataset is subsequently divided into training, validation and testing data. A symbol set with twenty-seven elements, A = {a…z, space}, is applied to simplify the dataset. The processing logic is shown as follows,

for *each symbol* $S_i \in$ *context C, where* $C = \{s_1 \ldots s_n\}$

if *'a'* < S_i < *'z'* ***then*** *write unary code to file*

else if *'A'* < S_i < *'Z'* ***then*** *convert to* $\{a, \ldots, z\}$ *and write unary code to file*

else *convert to blank and write unary code to file*

end for

```
%a b c d e f g h i j k l m n o p q r s t u v w x y z ''
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0        %'t'
 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0        %'h'
 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0        %'e'
```

First, all of the capital alphabets are converted to their corresponding lower case. The other symbols which do not belong to symbol set A are converted to space. Then each symbol is converted to a twenty seven length unary code. As shown in the example above, the word 'the' is successfully converted.

For this n-gram prediction, a three layer FTDNN network with 27 input neurons, 27 output neurons, extendible numbers of hidden layer neurons and extendible numbers of time delays is designed. Both input and output are encoded in unary code. A post processing function which ranks the twenty-seven output in a descending order has been used to produce the unary code results: the maximum value is converted into one and the rest of the values are converted into zeros.

3.3 Simulation with Plain Text Data Using Dataset One

In order to weight the computer simulation results, two concepts are introduced here, namely, Hitting Rate and First Three (FT) Hitting Rate. If given a testing metrics *P*, a target metrics *T* and a testing result metrics *R* with their numbers of lines and columns equal and expressed as *n, m* respectively, then the Hitting Rate is $HR = \{hr_i \mid hr_i = zeros(T - R_i)/n, i \in m\}$, where R_i is a vector of the ith Rank Conversion Value of *R*, and *zeros()* is the function to compute the numbers of zero vectors included in the metrics. For instance, the second Hitting Rate is the second best option for the symbol prediction in all output neurons of FTDNN, while the third Hitting Rate is the third best option etc. Obviously the sum of all Hitting Rates is $HR = \sum_{i=1}^{k} HR_i = 100\%$.

During the FTDNN model training and testing using the dataset one, the numbers of grams – [*1, 2, 3, 5, 7, 9, 11, 13*] which are represented by time delays, and the numbers of hidden neurons – [*1, 2, 3, 5, 7, 9, 15, 25, 50, 100*] are cross-designed and implemented. Thereinto, as the gram reaches *11* and the number of hidden neurons reaches *100*, or as the gram reaches *13* and the number of hidden neurons reaches *15* or onwards, the memory of current system is beyond its limit. Therefore, the computer simulation results are abandoned from *G-11 & H-100* onwards.

In order to demonstrate the effect of different grams on the hitting rate, a type of plots is produced based on the same computer simulation results, as shown in Figure 2. It has [*1, 2, 3, 5, 7, 9, 11*] grams associated with various number of hidden neurons. It is evident that *2, 3 & 5*-gram give the best three First Hitting Rates while *1, 2 & 3*-gram give the best three FT Hitting Rates (Note: in a small margin *2*-gram gives the best FT hitting rates and *3*-gram gives the best First hitting rate).

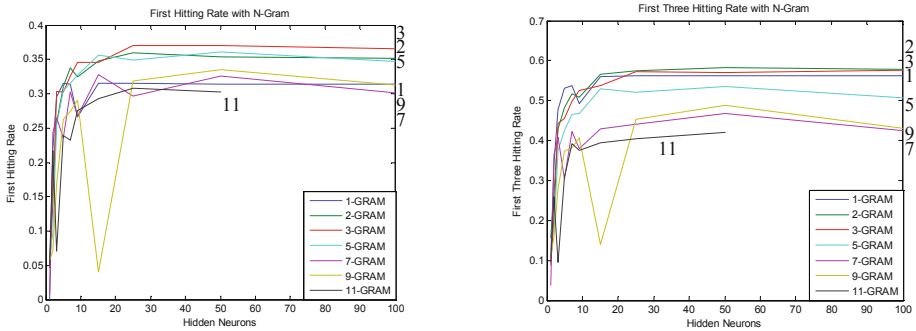

Fig. 2. N-gram First and First Three Hitting Rate curves; both have [1, 2, 3, 5, 7, 9, 11] grams associated with various number of hidden neurons

From both plots of Figure 2, the lower grams (*1, 2 & 3*) show a better convergence toward the maximum Hitting Rate (i.e. FT HR is around *56%*, First HR is around *33%*). Both figures illustrate that smaller Hitting Rates occur from *4*-gram onward. This proves that the more historical dataset input the more learning neural network space is needed, and the more training is needed toward convergence. Under the current training sample the results suggest that there is a best gram with certain number of hidden units to suit the prediction best. Beyond a critical point of prediction rate, further increase of gram or hidden unit doesn't help to achieve a better performance. Figure 2 also shows that the number of neurons in hidden layer affects the model's learning ability and Hitting Rate. For example, the number of neurons in hidden layer should not be too small to a structured symbol set {*a, … , z, space*} distribution; otherwise, it would be difficult for neural network to reach a good hitting rate. The *11*-gram testing stops at fifty neurons; a *27-100-27* three layer FTDNN model has failed to complete the training process under the current system environment.

3.4 N-Gram Prediction with Typing Data Using Dataset Two

As analyzed, the designed *n*-gram FTDNN models have shown that it can be applied to data prediction with a high capability. Here a user's typing data stream (i.e. dataset two) is used to further test the model. The users typing history is analyzed by the model to predict user's next typing intention. As the typing data stream is a typical noisy dataset which includes user's typing mistakes as well as self correction strokes such as symbols 'backspace' and 'delete', the model will not only learn the habits of user using language but also learn the self-correction actions which occurs in typing stream. For example, a self-correction action from a wrong typing word '*desj*' to the right word '*desk*' can be broken down in a typing data stream as,

d => e => s => j => backspace => k …

This is a typical adjacent-key-press error usually made by some people with motor disability or Parkinson disease. Through training, the FTDNN model is able to learn *2*-gram prediction rules between the predecessor and successor, for instance,

d → e
e → s

From the typing stream shown above, the model learns not only the existing noises such as 's' → 'j', but also the correction actions such as 'j' → 'backspace'. In practice, users just continue their typing without stop in spite of the mistakes, while the model should be able to correct the mistakes automatically or specify recommendations later on.

The collected data stream in dataset two is expressed in Virtual Key Codes [16]. In this research only editing virtual keys are adopted, other keys such as arrows are discarded. Then, the size of symbol set originally used by the model is extended into fifty three individual symbols, which apart from alphabet it also includes some other symbols such as,

VK_BACK	=>	*BACKSPACE key*
VK_RETURN	=>	*ENTER key*
VK_SHIFT	=>	*SHIFT key*
VK_DELETE	=>	*DEL key*

Based on the original design of FTDNN model, an extension to fifty-three units both at the input and output layer has been made. The dataset two has recorded both the key press 'down' status and 'up' status. Considering some disabled people specific typing behavior such as prolonged key press which would generate more 'down' keys corresponding to one 'up' keys, the keystrokes with 'down' status are selected by the pre-processing function for neural network training and testing. A comparison among the gram set [*1, 3, 5, 7, 9*] based on various numbers of hidden neurons - [*3, 5, 7, 9, 15, 25, 50, 100*] is shown in Figure 3. The first plot demonstrates a comparison of several grams' First Hitting Rates with an increase of hidden neurons. The second plot is a comparison of FT Hitting Rate between different grams.

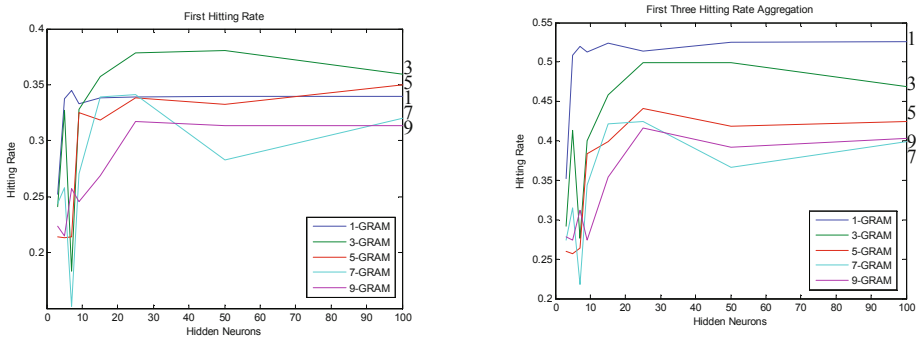

Fig. 3. [1, 3, 5, 7, 9] gram typing stream Hitting Rates; the first plot demonstrates a comparison of several grams' First Hitting Rates with an increase of hidden neurons. The second plot is a comparison of FT Hitting Rate between different grams.

Figure 3 shows that *1*-gram produces the maximum FT Hitting Rate of *53%* whereas *3*-gram with fifty hidden neurons produces the maximum First Hitting Rate of *38.1%*. Similar results have been obtained using dataset one; the lower grams (*1, 2 & 3*-gram) show a better solution using FTDNN model prediction under current circumstances. Both datasets demonstrate a highly accurate prediction rate (FT Hitting Rate, approximately *50%*) with FTDNN model.

4 Conclusion

This research work brings forth an original concept, Intelligent Keyboard hybrid framework for noisy language processing. It is developed as a hybrid solution based on FTDNN and n-gram technologies. Plain text dataset and user typing stream are tested in sequence based on extendable input and hidden neurons. Considering the user's typing history, a *38%* First Hitting Rate and a *53%* First Three Hitting Rate are obtained. The size of a training dataset, the occurrence of each symbol in a training dataset and the relationships among symbols of a training dataset all play an important role in the determination of a neural network language modeling prediction accuracy. The results further demonstrate that *1, 2 & 3*-gram generate better outcomes than other grams.

For future work, a distributed representation method to preprocess the typing symbols, where each symbol is represented by several features such as key distance, time stamp and symbols can be applied to the FTDNN models. In such a case, the prediction will not be solely based on the symbols themselves but also on related n-gram features.

Acknowledgments. The research is funded by Disability Essex [21] and Technology Strategy Board [22]. Thanks to Pete Collings for helpful advice and discussions.

References

1. Ouazzane, K., Li, J., Brouwer, M.: A hybrid framework towards the solution for people with disability effectively using computer keyboard. In: Proceedings of International Conference Intelligent Systems and Agents, Amsterdam, The Netherlands, pp. 209–212 (2008)
2. Trewin, S.: An invisible keyguard: Proceedings of ACM SIGACCESS ASSETS, Edinburgh, Scotland, pp. 143–149 (2002)
3. Bengio, Y., Ducharme, R., Vincent, P., Janvi, C.: A neural probabilistic language model. The Journal of Machine Learning Research 3, 1137–1155 (2003)
4. Schwenk, H., Gauvain, J.: Connectionist Language Modeling for Large Vocabulary Continuous Speech Recognition. In: Proceedings of ICASSP, Orlando, pp. 765–768 (2002)
5. Trewin, S., Pain, H.: A Model of Keyboard Configuration Requirements. In: Proceedings of International ACM Conference on Assistive Technologies, pp. 173–181 (1998)
6. The Dasher Project, Inference Group of Cambridge (November 14, 2007), `http://www.inference.phy.cam.ac.uk/dasher/` (accessed March 03, 2008)

7. Ward, J., Blackwell, A., et al.: Dasher - a Data Entry Interface Using Continuous Gestures and Language Model, http://www.inference.phy.cam.ac.uk/djw30/papers/uist2000.html (accessed March 03, 2008)
8. QWERTY (November 13, 2009), http://en.wikipedia.org/wiki/QWERTY (accessed November 13, 2009)
9. Prototype, n.d., http://www.sensorysoftware.com/prototype.html (accessed March 03, 2008)
10. Metaphone, Wikipedia (October 18, 2008), http://en.wikipedia.org/wiki/Metaphone (accessed January 23, 2009)
11. N-gram, Wikipedia (November 26, 2008), http://en.wikipedia.org/wiki/N-gram (accessed January 23, 2009)
12. Li, J., Hirst, H.: Semantic Knowledge in Word Completion. In: Proceedings of the 7th International ACM SIGACCESS Conference on Computers and Accessibility, pp. 121–128 (2005)
13. Li, J., Ouazzane, K., Jing, Y., Kazemian, H., Boyd, R.: Evolutionary ranking on multiple word correction algorithms using neural network approach. In: Palmer-Brown, D., Draganova, C., Pimenidis, E., Mouratidis, H. (eds.) EANN 2009. Communications in Computer and Information Science, vol. 43, pp. 409–418. Springer, Heidelberg (2009)
14. Focused Time-Delay Neural Network (newfftd), The MathWorks, http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/dynamic3.html (accessed January 23, 2009)
15. Haykin, S.: Neural Networks – A comprehensive Foundation, 2nd edn. Tom Robbins (1999)
16. Virtual key codes, http://api.farmanager.com/en/winapi/virtualkeycodes.html (accessed February 05, 2009)
17. Hardy, T.: Wikipedia (January 21, 2009), http://en.wikipedia.org/wiki/ (accessed January 22, 2009)
18. Cleary, J., Teahan, W.J., et al.: Unbounded length contexts for PPM. IEEE Computer Society Press, Los Alamitos (1995)
19. Bloom, C.: PPMZ–High Compression Markov Predictive Coder, http://www.cbloom.com/src/ppmz.html, ftp://ftp.cpsc.ucalgary.ca/pub/projects/text.compression.corpus/text.compression.corpus.tar.Z (accessed January 18, 2009)
20. Soukoreff, W., MacKenzie, S.: n.d. KeyCapture, http://dynamicnetservices.com/~will/academic/textinput/keycapture/ (accessed January 18, 2009)
21. Disability Essex, http://www.disabilityessex.org (accessed January 18, 2009)
22. Knowledge Transfer Partnership, http://www.ktponline.org.uk/ (accessed January 18, 2009)