Combining Sequence Alignment and the Structured Perceptron Framework to Increase the Throughput of Eye-Tracking Technology

Jesse Waite
Beiyu Lin

Abstract

This work details a method for fast inference of words from variable length sensor input streams derived from a low-cost eye tracking device. The method promises potentially faster and more error-tolerant methods for deriving user input from users whose communication is limited to eye-based communication interfaces and similar alternative and augmentative communication technology. The method uses a novel combination of sequence alignment and the structured perceptron framework to optimize the ranking of the target words over a lexicon consisting of tens of thousands of words (output classes). Our initial results show promising convergence and performance properties, and motivate the search for online methods.

Key words: Amyotrophic Lateral Sclerosis, eye-tracking technology, structured prediction

1. Introduction

Amyotrophic Lateral Sclerosis (ALS) is a neurodegenerative disorder by which patients lose motor function, often including the ability to speak. Often, their last means of communication is via eye-based communication interfaces (EBCI) that interpret eye-gaze sensor input as discrete key presses on a virtual keyboard via trigger-based key activation schemes [3]. Historically these interfaces are costly, somewhat slow throughput, and incorporate few machine-learning features to make them faster, error tolerant, and more usable [2]. The overall cost and usability constraints prevent many patients with neurodegenerative disorders and other physically-degenerative conditions from communicating at a natural conversational rate.

A key observation leveraged in this work is that the geometric and statistical features of eye traces, as well as the geometric and linguistic properties of target words, encode significant information. By omitting such features, the model of discrete key activations (DKA) arguably imposes an over-constrained and unnecessarily slow method of user input. A common tactic to overcome the DKA bottleneck is to leverage language models incorporating word completion or next-word prediction, thus helping users output words at a faster rate. This approach suffers as the low expected accuracy of word prediction results in low expected keystroke reduction, and likewise the approach does not resolve the prior constraints of the DKA model.

In this paper, we proposed a novel approach to directly infer language by leveraging the complete combination of statistical, geometric, and language models of words and their traces. This approach is more promising, circumvents the bottleneck of the discrete input model by directly inferring words from sensor data, and offers better error tolerance from misspellings and sensor error. Further, the approach is easily augmented by word prediction and completion features to perform as well or better than the DKA model. In short, by leveraging the complete

joint structural characteristics of sensor input streams and language outputs, faster and more usable EBCI's can be devised.

In our experiments, an eye tracking device tracks the user's eye-focus position on a virtual keyboard interface. The device determines the user's focus position to within an inch or less variance, and sets the mouse cursor to that position. In this way, the cursor tracks the user's eye position on the interface, generating a stream of (x,y) points for a given word trace, often at a resolution of 30-50Hz.

Accordingly, we define a variable-length sensor input stream, or 'trace', **x** of screen coordinate (x,y) pairs as the primary input by which to perform inference. We denote vectors **y** as the target output that represents the character-wise point sequence for some word *w* in lexicon **L**. Compared with traditional structured prediction frameworks, the primary obstacle is that both the input **x** and the output **y** vary in length. One must either use a dynamic method to optimize the mapping of variable length **x** and **y** training inputs, or produce a feature transformation that reduces variable length inputs to a fixed-length feature vector. The former is the subject of this paper, and the latter, though especially promising for online methods, is left to future work.

2. Signal Properties and Processing Strategies

A key contribution of this project is the construction of word trace distance minimization, and the distance metrics used to drive it. Intuitively, the highest quality features for inference obtain minimal distance with respect to the target word, while maximizing the distance to dissimilar words. Regarding these features, it is important to foreground the signal characteristics that enable direct inference. The most important observation is that traces have strong geometric and statistical relationships with the underlying sequence of points defined by the character-wise key positions of their corresponding target word.

The simplest and perhaps most effective such metric is to minimize the distance over all points **x** with respect to the point sequence **y** for some word *w* in **L**:

$$dist_w(\boldsymbol{y}_w, \boldsymbol{x}) = min \sum_{i}^{|\boldsymbol{y}_w|} \sum_{j}^{k_i} distance(y_i, x_i)$$

Equation 0: Recursive Trace Distance Minimization

This metric uses dynamic programming operator *min* uses simplified sequence alignment methods to assign sensor inputs to their nearest key point y_i in word *w*, and sums over the distance given by this minimum assignment. If *distance* is the Euclidean distance (L2-norm), this metric is highly effective for ranking target words via geometric distance, and only suffers due to differences in the lengths of ***y*** or **x**, causing a bias toward shorter words. However, the *min* distance construction will prove useful, and is detailed later.

The input signal itself embeds characteristics reflecting events such as likely key triggers, key-press holds, and between-key transitions. These can also be included in the *distance* metric to overcome the issues of using only the L2-norm. As a demonstration, note the regular features of the graphs in figure 1 through 5, for the user-generated **x** sensor stream of (x,y) coordinates for the word "Biloxi". The stream was generated with minimal user dwell time per character.

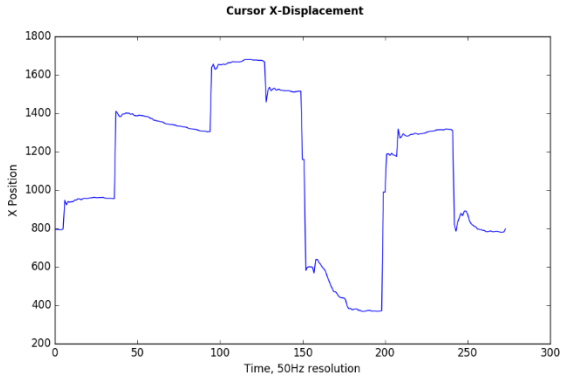

Figure 1

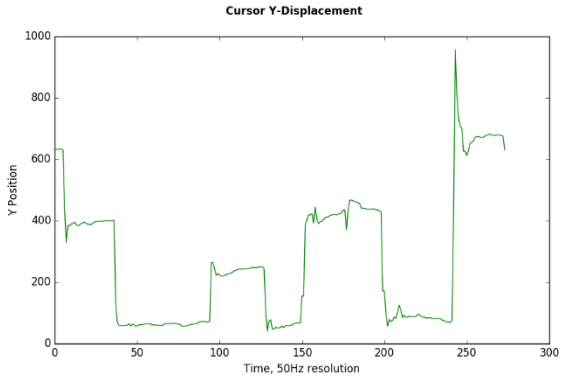

Figure 2

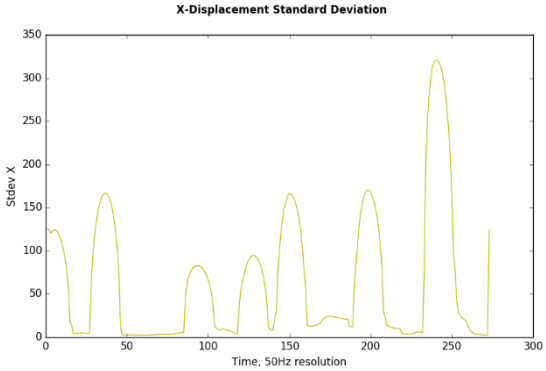

Figure 3

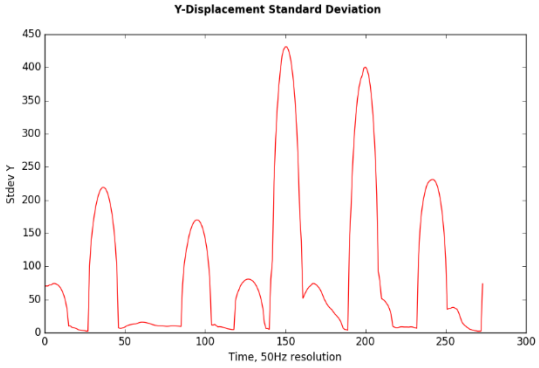

Figure 4

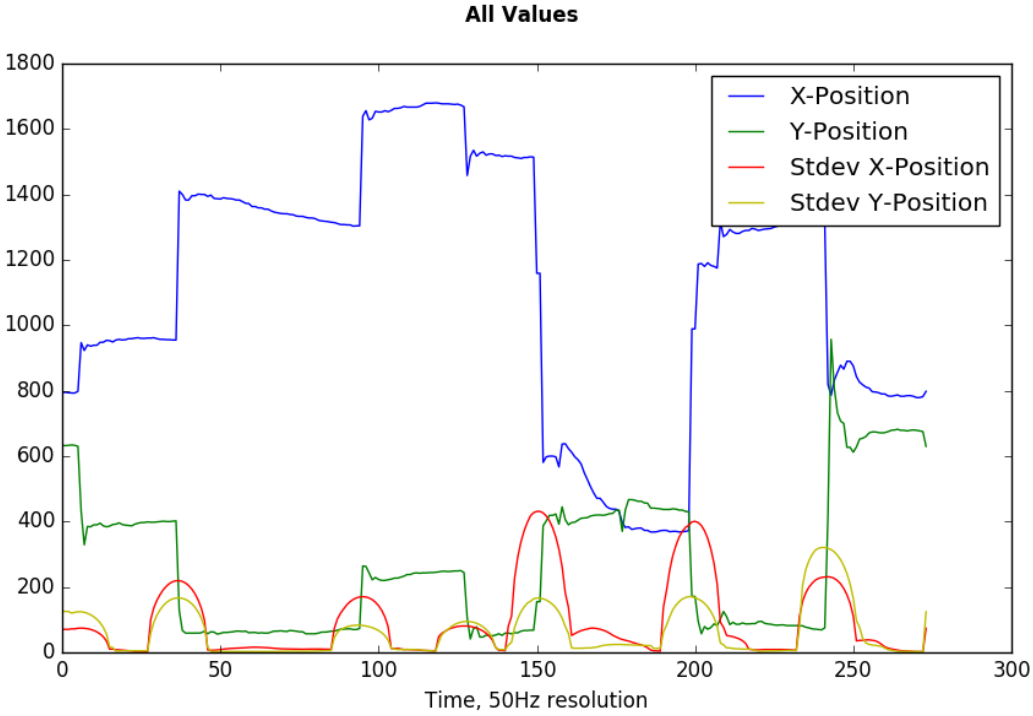

Figure 5

Notably, the signal features in terms of vertical deviations are strongly correlated among all four graphs for time slices on the x-axis (figure 5). For instance, when the user's eye manipulates the cursor over the keyboard interface, the displacement plots of the x and y cursor position show an apparent step-like quality: vertical for between-key transitions, and nearly horizontal when focusing on a particular key (figures 1 and 2). Likewise, for each of these step features, the standard deviation along any single coordinate axis over some small window of sensor readings shows spiking behavior during inter-key cursor transitions, but falls near zero when the user focuses on a single point (figures 3 and 4).

These properties are necessary, but not sufficient to identify key focus states and inter-key transitions. From the perspective of HMM or CRF modelling, these are signal properties that could model latent key triggers using these emission characteristics to devise a simple character decoder [0,4]. Such a local model could then be used to estimate word probabilities, for instance using the forward algorithm for HMM's [6]. The drawback is that such models do not search in the output space of complete words and their geometric or linguistic properties, representing significant loss of information. Our method instead operates over both local signal event features and the complete output space using a non-parametric method, making it highly extensible to a wider range of problems.

Given the signal characteristics and geometric distance constructions from prior, we desire to use the comprehensive set of local and structured output features to define a weighting by which to assign, and subsequently score, an assignment of edges in the bipartite mapping between the signal points and some target word, as shown:

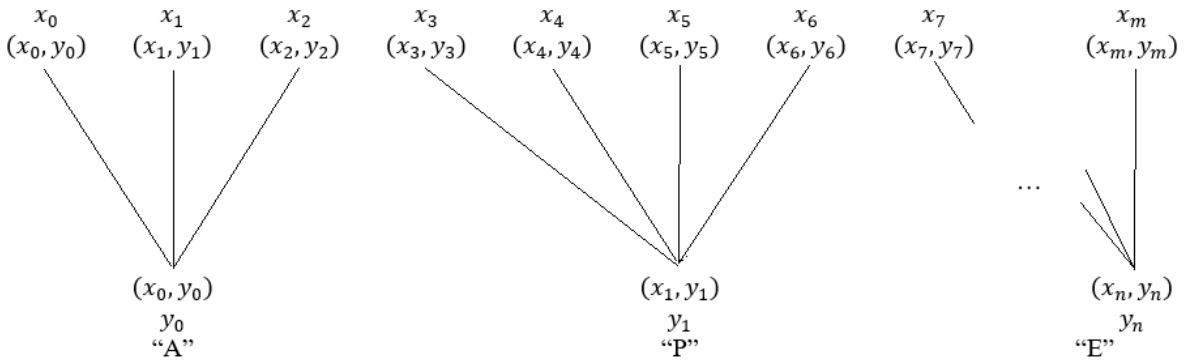

Figure 6: An example assignment of sensor inputs to the truncated candidate output *w* 'APPLE'

The problem reduces to learning a set of edge weights denoted by vector $\boldsymbol{\theta}$, by which a bipartite mapping can be obtained which minimizes the distance between a signal stream and the character-wise point-sequence of target words denoted by vector **y**. Given the edge weights $\boldsymbol{\theta}$, dynamic programming is suitable to discover the optimal bipartite assignment of edges between **x** and **y**. The larger problem addressed in this work is that of optimizing these weights over all words, such that the target output word *w* is ranked highest among all candidate words in **L**, on average.

3. Proposed Solution

The requirement for optimizing a set of variable-length inputs and outputs immediately suggests dynamic programming, coupling the recursive properties of sequence alignment with a parameterized distance metric that can be optimized via limited training examples. In contrast with standard sequence alignment, which typically uses fixed and manually-defined scoring weights, we optimize the scoring weights $\boldsymbol{\theta}$ by performing supervised learning.

Given weights $\boldsymbol{\theta}$, the prediction problem is constructed as in Algorithm 1:

Algorithm 1: Prediction

Input: A finite, variable-length sensor stream x of (x,y) sensor readings, distance metric weights w, and lexicon L

Output: A word $w \in L$, as represented by its character-wise coordinates y_w^*.

candidates = {};
For $y_w \in L$**;**
    $score_w = Align(\boldsymbol{x}, \boldsymbol{y_w}, \boldsymbol{\theta})$**;**
    $candidates = candidates \cup (w, score_w)$**;**
return $argmax_{w \in \boldsymbol{L}}(candidates)$**;**

Our primary contribution is a training method for optimizing the weights $\boldsymbol{\theta}$ by which the ranking of the correct output is maximized over all other words in lexicon **L**. For perspective, our English lexicon contains around 60,000 words, so the inference method operates over a large output space of tens of thousands of possible classes. The ranking of the target output *w* is maximized using geometric and statistical distance metrics to drive the alignment procedure, combined with a simple perceptron wrapper as shown in algorithm 2. In our work, we successfully used the basic perceptron definition, but one could as easily use the averaged perceptron, or return the weights $\boldsymbol{\theta}^*$ determined by a validation set.

Algorithm 2: Training

Until maxIterations or convergence;
For $(\boldsymbol{x}, \boldsymbol{y}) \in D$, $(iterate\ \ training\ \ examples)$**;**
    $\hat{w} = Predict(\boldsymbol{x}, \boldsymbol{L}, \boldsymbol{\theta})$ $(from\ \ Algorithm1)$**;**
    $\hat{y}$ **=** $WordToPointSequence(\hat{w})$ **;**
    if $\hat{\boldsymbol{y}} \neq \boldsymbol{y}^*$**:** $(weight\ \ update)$**;**
        $\boldsymbol{\theta} = \boldsymbol{\theta} + \alpha(\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y}^*) - \boldsymbol{\Phi}(\boldsymbol{x}, \hat{\boldsymbol{y}}))$**;**

Algorithm 3: Feature Function $\boldsymbol{\Phi}(x, y, \theta)$

$Align(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{\theta})$, *(Call Align to initialize backpointers);*
$\boldsymbol{\Phi_{x,y}} = BackTrack()\{$ **;**
    $\boldsymbol{\Phi_{x,y}} = 0$**;**
    $row = |\mathrm{x}| - 1$**;**
    $col = |\mathrm{y}| - 1$**;**
    Until row == 0 and col == 0 ;
        $\boldsymbol{\Phi_{x,y}} = \boldsymbol{\Phi_{x,y}} + \phi_{row,col}$, $(Sum\ features\ of\ the\ cell\ at\ row,\ col)$ **;**
        $row, col = FollowPointer(row, col)$**;**
    };
return: $\boldsymbol{\Phi_{x,y}}$**;**

The *Align* method is perhaps the most important but possibly opaque component of this method, calling for a concrete example. *Align* iterates over all (x,y) coordinate inputs in **x**, comparing each of these via some parameterized distance metric with each (x,y) point in **y**, where **y** is the sequence of points representing some word. *Align* recursively calculates the minimum distance of the minimum alignment of the **x** and **y** point sequences, using a standard dynamic programming recursion. Notably, this backtracking defines only two cases for this domain, UP and LEFT, which proved sufficient for this application, although a diagonal DIAG move could be added for others. *Align* maintains the back-pointers of the optimal alignment between these two sequences. Summing over the $\boldsymbol{\phi}$ components along this path gives the feature function $\boldsymbol{\Phi}(\boldsymbol{x}, \boldsymbol{y})$, as shown in ***Backtrack***.

Algorithm 4: Alignment Procedure $Align\ (x, y, \theta)$

initialize dp **table with** $|x|$ **rows, and** $|y|$ **columns of cells containing** $score_{i,j}$ **and** $backpointer_{i,j}$**;**
initialize first row;
$score_{0,j} = \Phi(0, j) \cdot \theta, backpointer_{0,j} = LEFT, for\ j \in |y|$**;**
initialize first column;
$score_{i,0} = \Phi(i, 0) \cdot \theta, backpointer_{i,0} = LEFT, for\ i \in |x|$**;**
recursion: For all $i \in |x|, Forall j \in |y|$**;**
$score_{i,j} = max\{score_{i-1,j} + \Phi_{i,j}\} \cdot \theta_{up}, score_{i,j-1} + \Phi_{i,j} \cdot \theta_{left}$**;**
$backpointer_{i,j} = argmax_{UP,LEFT}\{\cdot\}$ $(pointer\ set\ to\ direction\ of\ max\ from\ previous\ line)$**;**

For illustrative purposes, the simplest form of $\boldsymbol{\phi}$ consists only of the Euclidean distance between a point in **x** and some point in **y**. This construction yields a score representing the minimum pointwise distance between sensor inputs and points in a character sequence on the interface, via the optimal sequential alignment between **x** and **y**. In practice, other useful properties can be added to $\boldsymbol{\phi}$: local signal properties of **x** such as local standard deviation, or local properties of **y**, such as character-gram transition probabilities. The expressive power of this linear feature distance metric allows the practitioner to engineer meaningful signal attributes for their application, the hidden states one is trying to model (here, discrete key events), and the target outputs. In a sense, the local features define input, output, and joint x/y features, whereas the backpointer directions along the dynamic program can represent the hidden states one wishes to model, such as key triggers, key-holds, and inter-key transitions. Over training iterations, the method converges to the parameters by which an arbitrary signal can be mapped onto a target output.

In short, distance minimization occurs over signal properties and output symbols via a dynamic program, by which a score can be assigned to every word in some lexicon **L**. The rank-discrepancy, if any, between the target output and the correct output provides the error signal by which to modify the training weights. Although the distance metrics and the alignment method are highly novel, the method is preceded by highly similar work combining sequence alignment and the structured perceptron performed by [4]. Likewise, procedures for optimizing sequence alignment parameters goes back to [7] and the predecessors cited in that work.

4. Experimental Evaluation

To test the proposed solution, we relied on only 13 self-generated example inputs, and a lexicon of approximately 60,000 English words. The sparsity of examples is due to the absence of EBCI datasets consisting of user target words with corresponding eye-sensor input streams. Even this small handful of training examples was sufficient for satisfactory performance, as detailed later. The sensor streams were derived using a 2014 EyeTribe eye sensor operating at 50Hz, situated approximately two feet from a user gazing on a 20" monitor.

The local features within the *Align* method were Euclidean distance between points x_i and y_j, $\sigma_{x_{x,k}}$, and $\sigma_{x_{y,k}}$, where $\sigma_{*,k}$ denotes the standard deviation of the input signal **x** over a window of *k* points along either the x or y coordinate axes. In these experiments, *k* was arbitrarily set to 19 via manual observation, but should be determined by the sensor resolution and precision. The learning rate was set to α = 1E-6, which is somewhat small due to the large magnitude of the features.

To evaluate our method, we tracked the decline in the average ranking of the target word over training iterations, where a rank of zero represents the highest scoring word, hence a lower target word rank is desirable. The training error was tracked by measuring the average ranking of the correct output over the entire dataset for a single iteration of training. We found this was the only way to empirically measure algorithmic performance and convergence, since the objective is to reduce the ranking of the target output, and the score is just a relative ranking scheme. Schemes involving the score of the target output were not stable, since small changes in the weights can dramatically alter the magnitude of the scores, and only a score's value relative to another is truly important, not its scalar value.

5. Results

As shown below, the algorithm converged rapidly within only a few iterations. Such rapid convergence suggests overfitting so few training examples. However, this outcome is more likely because the chosen distance metrics perform most of the labor of inference. In other words, the trace minimization of *Equation 0* lends itself to a straightforward, analytic method of inference; wrapping this method with perceptron-based parameter optimization simply maximizes the accuracy of an already mostly-accurate method of inference.

The sawtooth pattern after initial convergence is also a notable feature of the results. The pattern is not surprising, since the rank-based error signal of the perceptron is the non-maximal ranking of the target word. Near convergence, small changes in the weights cause some targets to lose the top-most ranking, the correction of which propagates non-maximal rankings elsewhere, and so on, creating this oscillation around local optima. The remedy for this would be to introduce measures for relaxing the strict constraint that the target be the topmost ranked word over such a large vocabulary. The error signal could be relaxed to require only that the top word occur within some window of top-k rankings, or the error signal could be scaled by the difference in the rankings of the target word and the actual prediction. Likewise, an averaged perceptron could be used, or one could simply snapshot the best performing weights $\boldsymbol{\theta}$ tested on a validation set.

6. Conclusions

The results provide empirical justification of the convergence of this signal inference method, using a novel combination of dynamic programming and the structured perceptron. The results show a high degree of accuracy, and promise faster communication throughput for EBCI users. The benefit of this method is its extensibility to a range of inferential signal processing methods, and its promise for other interfaces and problems. A primary drawback of the approach is that the inference method must score all words in the lexicon using an $O(n^2)$ dynamic programming procedure, giving overall performance of $|\boldsymbol{L}| * O(n^2)$. Since English language words are short on average, the term n^2 reduces to $\sim cn$ for some small c representing average word length (about 5 characters, [1]), and the method is surprisingly faster than expected. Further, the inference burden can be mitigated by through early termination of distance calculation for very poorly scoring candidate words.

The most significant drawback is that the method operates over complete input streams, and hence cannot readily perform online word inference (completion) for partial input: all input must be received before the inference method can begin. This is tolerable if the method gives faster user communication throughput times overall, and if the inference procedure is optimized as mentioned previously, but warrants a notable criticism by online methods, if any can be found.

Lastly, this work offers no formal proof of the method's convergence properties, which likely varies according to the construction of features and backtracking representation. For instance, it would be possible to mistakenly define non-convex distance features, such that the algorithm would not converge. In general, given the non-convex space of rankings over words, the best that can be expected is that the algorithm converges to a point at which it begins to oscillate (chatter) around this space, correcting one error whilst generating others, and so on. The remedies mentioned in the previous section are palliatives, and do not resolve a problem resulting from the score-and-rank framework.

7. Future Work

Incorporating longer range features into the perceptron would greatly enhance the performance of the algorithm in the target application, but were omitted to focus on the core method. Nonetheless, the method is amenable to the addition of both local signal properties and arbitrary longer range language features, such as n-gram models. These trivial feature additions would improve performance in the domain, but are not an algorithmic contribution.

The most promising area future work is developing online methods to run such an inference algorithm, or to identify a reduction to a standard structured prediction framework. One way to do is to represent input not as a variable length stream of inputs, but rather as an OCR problem for which sensor inputs on the interface are treated as binary pixels within a character stroke. By modeling the interface as a binary or integer based map, the input obtains a fixed length representation, and off the shelf structured prediction methods could be tested, many of which offer online inference. Additionally, just like the method described in this paper, this method could easily incorporate local and long-range features of the joint input-output space: local and statistical signal properties, joint input-output geometry, language models, and so on.

The drawback of this approach is that it discards the sequential nature of sensor input; however, this might also be modelled through various features. A more serious drawback is that, like OCR, this approach requires sensor stream training examples for every word in a lexicon—an enormous number of classes. This could be remedied by writing a generator to simulate training examples according to the expected noise parameters of the eye sensor. In short, by modelling **x** not via variable input streams, but via fixed-length interface representations, the problem reduces to standard structured prediction methods.

Acknowledgments

We would like to thank Team Gleason for the inspiration for this project, and Professor Jana Doppa, for challenging us to solve such applied problems using structured prediction.

Works Cited

[0] Baum, Leonard E., and Ted Petrie. "Statistical inference for probabilistic functions of finite state Markov chains." *The annals of mathematical statistics* 37.6 (1966): 1554-1563.

[1] Bochkarev, Vladimir V., Anna V. Shevlyakova, and Valery Solovyev. "Average word length dynamics as indicator of cultural changes in society." *CoRR, abs/1208.6109* (2012).

[2] Calvo, Andrea, et al. "Eye tracking impact on quality-of-life of ALS patients." *Computers Helping People with Special Needs* (2008): 70-77.

[3] Duchowski, Andrew T. "A breadth-first survey of eye-tracking applications." *Behavior Research Methods, Instruments, & Computers* 34.4 (2002): 455-470.

[4] Freitag, Dayne, and Shahram Khadivi. "A Sequence Alignment Model Based on the Averaged Perceptron." *EMNLP-CoNLL*. 2007.

[5] Lafferty, John, Andrew McCallum, and Fernando Pereira. "Conditional random fields: Probabilistic models for segmenting and labeling sequence data." *Proceedings of the eighteenth international conference on machine learning, ICML*. Vol. 1. 2001.

[6] Rabiner, Lawrence, and B. Juang. "An introduction to hidden Markov models." *IEEE ASSP Magazine* 3.1 (1986): 4-16.

[7] Ristad, Eric Sven, and Peter N. Yianilos. "Learning string-edit distance." *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20.5 (1998): 522-532.