# INCREMENTALLY LEARN THE RELEVANCE OF WORDS IN A DICTIONARY FOR SPOKEN LANGUAGE ACQUISITION

Vincent Renkens[1], Vikrant Tomar[2], Hugo Van hamme[1]

[1]Department Electrical Engineering-ESAT, Katholieke Universiteit Leuven
Kasteelpark Arenberg 10, Bus 2441, B-3001 Leuven Belgium
[2]Fluent.ai, 780 Avenue Brewster, QC H4C 2K1 Montréal Canada
{vincent.renkens,hugo.vanhamme}@esat.kuleuven.be, vikrant@fluent.ai

## ABSTRACT

This paper discusses a spoken language acquisition system for a command-and-control interface. The proposed system learns a set of words through coupled commands and demonstrations. The user can teach the system a new command by demonstrating the uttered command through an alternative interface. With these coupled commands and demonstrations, the system can learn the acoustic representations of the used words coupled with the meaning or semantics. In previous work the focus was mainly on a batch learning scheme to train the model. All the commands and demonstrations had to be stored and the model had to be retrained from scratch every time a new demonstration was given by the user. This work presents a Bayesian learning scheme where the dictionary of learned words can be updated when new data is presented. The dictionary can automatically expand to add new words or shrink to forget old words. The proposed system is tested on a language acquisition task where the user suddenly starts using new words. The results show that the proposed system can learn the new words quicker than a baseline where the size of the dictionary cannot be adjusted.

***Index Terms*—** Vocabulary learning, Spoken language acquisition, Non-negative Matrix Factorisation, Machine learning, Bayesian methods

## 1. INTRODUCTION

Designing a natural user interface for command and control applications is a challenging task. The designer has to make hard choices, for example, how to name objects and actions. It is impossible for the designer to make choices that satisfy all potential users, so the user has to yield to the designer's choice. It would be more desirable if the technology could adapt to the user instead of the other way around.

We propose an interface that learns its vocabulary from the user. The user can teach the system by uttering a command and subsequently demonstrating the command through an alternative interface. From the demonstration, the important concepts, called semantics, are extracted and the system is trained to link spoken words to these concepts. The system is trained completely speaker and application dependent. This means that the number of words the system has to learn is very limited. For the applications where this system can be used the vocabulary size is limited to a couple dozen words. However, the amount of training data is also very limited, because demonstrating a command requires effort from the user which should be minimised. To summarise, the system will be able to learn a limited set of words with as little training data as possible. Potential applications for such an interface include domotics [1], smartphones/watches, automotive or robotics [2].

In the past we have proposed a spoken language acquisition system [3, 4, 5] that learns a dictionary of spoken words with Non-negative Matrix Factorisation (NMF) [6]. NMF can find recurring acoustic patterns (like words) and link them to the semantics by separating the contributions of the words in the speech signal. The semantics of an unknown command can then be found by detecting which of the learned words occur in the utterance. Other applications of NMF include music analysis [7], bioinformatics [8] and image classification [9].

Our previously proposed language acquisition system faces two problems that are tackled in this paper. First, the number of words in the dictionary has to be defined beforehand. In previous work the dictionary size was determined by considering how many unique concepts were extracted from all user demonstration plus some extra words that are not linked with a meaning (e.g. "please"), called garbage words. This approach has two problems, (1) this does not consider a scenario where the user uses several words for the same concept and (2) The number of garbage words has to be guessed, which can be difficult.

The second problem with the language acquisition system is that it is a batch learning algorithm. The system should be able to learn on the fly and the user should be able to use it while it is still learning. To allow this kind of behaviour all

Research funded by a PhD grant of the Research Foundation Flanders (FWO)

past commands have to be remembered and the model has to be trained again with every update. This is inefficient both from a memory and a computational view.

Both of these problems have been tackled in the past separately. In [10] a technique for automatically learning the relevance of words was proposed. The dictionary can be initialised with a large number of words and at the end of training only the relevant words are kept. This technique is called Automatic Relevance Determination (ARD). In [11] The NMF batch algorithm was adapted to an incremental algorithm. The main contribution of this paper is combining the techniques to solve both presented issues. The system will be able to incrementally learn a set of words and their relevance. To make this possible both techniques have to be adjusted.

The remainder of this paper is organised as follows. In section 2 the spoken language acquisition system that we have proposed in the past is explained. In section 3 the main contribution of this paper, the technique that can incrementally learn the relevance of words is presented. In section 4 the experimental results will be presented and discussed. Finally, some conclusions will be formulated in section 5.

## 2. LEARNING A SET OF WORDS WITH NMF

Spoken language acquisition based on limited amount of commands and demonstrations is a non-standard problem, so non-standard techniques are used to tackle it. For the ease of the reader the spoken language acquisition system that we have been using in our previous work [3, 4, 5] is explained in this section.

### 2.1. Problem statement

The goal of the system is to learn a set of words based on vocal commands, called a dictionary, and link them to a meaning based on the corresponding demonstrations. No transcriptions are available. What *can* be extracted from the demonstration is a set of "concepts" that are relevant for the demonstration. This set of concepts is called the semantics of the command. For example, if the user gives a demonstration of calling a person, the concepts of calling and that person can be extracted. It can then be assumed that there will be a word (or a set of words) in the vocal command that can be linked to these concepts. The vocal command for the example could be "Call John Doe". "call" can be linked to the concept of calling and "John Doe" can be linked to the concept of the person that is being called.

From the commands and corresponding demonstrations we can extract a collection of $N$ pairs $\{(\boldsymbol{v}_n^{\text{s}}, \boldsymbol{O}_n), n \in [1 : N]\}$. $\boldsymbol{v}_n^{\text{s}}$ is a vector representing the semantics of the $n^{\text{th}}$ command. Every element in this vector corresponds to a concept, if the concept was extracted from the demonstration it is set to 1 or active and 0 or inactive otherwise. $\boldsymbol{O}_n$ is called the observation matrix and is a $D \times T_n$ matrix of acoustic features representing the $n^{\text{th}}$ spoken command. $D$ is the dimension of the feature vector and $T_n$ is the number of acoustic frames.

From the semantic-observation pairs a set of acoustic word-like patterns will be learned that can be linked to the semantics. The semantics of an unknown command can then be discovered by determining which of these word-like patterns occur in the vocal command. In the remainder of this paper we will simply refer to "word-like patterns" as "words".

### 2.2. A spoken command as a combination of words

To learn the set of words, a vocal command will be considered to be a weighted sum of words. For this purpose the observation matrix of an utterance $\boldsymbol{O}$ will be converted to a vector representation $\boldsymbol{v}^{\text{a}}$. This vector represents the entire utterance, how it is computed will be discussed at the end of this section. This acoustic representation is then approximated as:

$$\boldsymbol{v}^{\text{a}} \approx \sum_{k=1}^{K} h_k \boldsymbol{w}_k^{\text{a}} \tag{1}$$

$\boldsymbol{w}_k^{\text{a}}$ is the acoustic representation of the $k^{\text{th}}$ word in the dictionary and there's a total of $K$ words. $h_k$ is the weight of the $k^{\text{th}}$ word in the sum. These weight will be called the activations. For example, in the command "Call John Doe", the words "call" and "John Doe" will be activated. The acoustic representations will be constrained to be non-negative to avoid words cancelling each other out.

The same approximation can be made in the semantics:

$$\boldsymbol{v}^{\text{s}} \approx \sum_{k=1}^{K} h_k \boldsymbol{w}_k^{\text{s}} \tag{2}$$

$\boldsymbol{w}_k^{\text{s}}$ is the semantic representation of the $k^{\text{th}}$ word in the dictionary. Notice that the activations for the acoustic representation and the semantics are shared, making sure that the semantic representation is linked to the acoustic representation.

### 2.3. Finding word representations with NMF

To find the acoustic and semantic word representations, the acoustic and semantic representations of all commands are combined into an acoustic data matrix $\boldsymbol{V}^{\text{a}}$ and a semantic data matrix $\boldsymbol{V}^{\text{s}}$. By stacking these data matrices on top of each other into the data matrix $\boldsymbol{V}$ we can write the acoustic and semantic decompositions from equations (1) and (2) for all commands into one matrix decomposition:

$$\begin{bmatrix} \boldsymbol{V}^{\text{s}} \\ \boldsymbol{V}^{\text{a}} \end{bmatrix} \approx \begin{bmatrix} \boldsymbol{W}^{\text{s}} \\ \boldsymbol{W}^{\text{a}} \end{bmatrix} \boldsymbol{H} \tag{3}$$

$$\boldsymbol{V} \approx \boldsymbol{W}\boldsymbol{H} \tag{4}$$

$\boldsymbol{W}^{\text{a}}$ and $\boldsymbol{W}^{\text{s}}$ contain the acoustic and semantic representations of the words, they are called the acoustic and semantic

dictionary (stacked into the dictionary $\boldsymbol{W}$). $\boldsymbol{H}$ contains the activations of the words in all commands and is called the activation matrix.

The dictionary and activation matrix are found with NMF [6]. NMF decomposes a non-negative matrix into 2 non-negative matrices, so it's exactly what is needed to find the word representations. During training NMF is used to find the acoustic and semantic dictionary. To find the semantics of an unknown command during testing the acoustic dictionary is used to find word activations in the spoken command. Subsequently the semantic representation can be computed by multiplying the semantic dictionary with these activations.

### 2.4. Acoustic representation of a spoken command

There are 3 main requirements for the acoustic representation. (1) the vector has to have a fixed length so they can be used as columns in a matrix, (2) The vector needs to be non-negative in order for it to be usable in NMF and (3) the contributions of the words need to be additive so equation (1) holds. A representation that meets all of these requirements is to use counts of acoustic events. In our previous work [3, 4, 5] these acoustic events were found by fitting a Gaussian Mixture Model to the data, each Gaussian was assumed to be one acoustic event. In this paper phones are used as acoustic events. A Deep Neural Network (DNN) [12] is trained on a large database of speech and then used to extract phone posteriors for the spoken command.

The downside of just counting phones in the input speech is that all timing information of when the phones occur in the utterance is lost. This is somewhat alleviated by counting phone *co*-occurrences instead. The acoustic representation is a count of the co-occurrence of two phones with a fixed delay between them. Several delays are used and stacked to capture multiple time spans. This acoustic representation is called a Histogram of Acoustic Co-occurrences (HAC) [13].

## 3. INCREMENTAL AUTOMATIC RELEVANCE DETERMINATION

In this section the main contribution of this paper will be presented. The two problems with the current language acquisition system that were presented in the introduction will be tackled. Automatically determining the size of the dictionary will be done with ARD, presented in [10]. Updating the dictionary when a new batch of data is presented will be done with Incremental NMF presented in [11]. First the proposed technique is explained and at the end the differences with the techniques from [10] and [11] will be highlighted.

### 3.1. Model

In the NMF learning algorithm proposed by Lee and Seung [6], the matrix factors are learned by maximising the log likelihood. Two main changes will be made to the objective to make both ARD and incremental learning possible. (1) for every word a relevance parameter ($\boldsymbol{\nu}$) representing how often the word is activated will be added and (2) instead of the log likelihood the log posterior will be maximised:

$$(\boldsymbol{W}, \boldsymbol{H}, \boldsymbol{\nu}) = \underset{(\boldsymbol{W},\boldsymbol{H},\boldsymbol{\nu})}{\arg\max} \log P(\boldsymbol{W}, \boldsymbol{H}, \boldsymbol{\nu}|\boldsymbol{V}) \tag{5}$$

Remember that $\boldsymbol{V}$ is the data matrix that contains both the semantic representations of the commands and the HACs. With Bayes' rule we can write the posterior probability in terms of likelihood and priors:

$$P(\boldsymbol{W}, \boldsymbol{H}, \boldsymbol{\nu}|\boldsymbol{V}) = \frac{P(\boldsymbol{V}|\boldsymbol{W}, \boldsymbol{H})P(\boldsymbol{W})P(\boldsymbol{H}|\boldsymbol{\nu})P(\boldsymbol{\nu})}{P(\boldsymbol{V})} \tag{6}$$

Every factor in the posterior plays a well defined role in the learning algorithm (except for $P(\boldsymbol{V})$ which is constant). The likelihood $P(\boldsymbol{V}|\boldsymbol{W}, \boldsymbol{H})$ determines how well the matrix factors fit the data. A Poisson distribution is used as the likelihood function because it's appropriate for count data, which is used to construct the data matrix:

$$\log P(\boldsymbol{V}|\boldsymbol{W}, \boldsymbol{H}) \stackrel{c}{=} \sum_{f=1}^{F}\sum_{n=1}^{N} [V_{fn} \log(WH)_{fn} - (WH)_{fn}] \tag{7}$$

$\stackrel{c}{=}$ is equality up to a constant and $F$ is the number of rows in the data matrix.

The activation prior $P(\boldsymbol{H}|\boldsymbol{\nu})$ pushes the algorithm to only activate words that are relevant, so the prior will push the activations to 0 for irrelevant words. An exponential prior with the relevance parameters as the scale parameters is used to obtain this effect:

$$\log P(\boldsymbol{H}|\boldsymbol{\nu}) \stackrel{c}{=} -\sum_{k=1}^{K}\sum_{n=1}^{N} \left[\frac{H_{kn}}{\nu_k} + \frac{1}{\nu_k}\right] \tag{8}$$

The dictionary prior $P(\boldsymbol{W})$ pushes the algorithm not to deviate too far from the dictionary of the previous model update. This means that the dictionary is remembered through the dictionary prior. A gamma distribution could be used for the dictionary prior because it is conjugate to the Poisson distribution. However, NMF is scale invariant. This means that when a row in $\boldsymbol{H}$ is scaled and a column in $\boldsymbol{W}$ is inversely scaled the same result is obtained. This means that when the activations of a word are pushed to 0, this could be compensated by scaling up the word representation, which is not the desired behaviour. To avoid the scaling ambiguity the word representations are normalised so they sum to 1. When the variables of the gamma distribution are normalised, a Dirichlet distribution is obtained:

$$\log P(\boldsymbol{W}) \stackrel{c}{=} \sum_{k=1}^{K}\sum_{f=1}^{F} \alpha_{fk} \log W_{fk} \tag{9}$$

$\boldsymbol{\alpha}$ are the shape parameters of the Dirichlet distribution and the sufficient statictics of the dictionary. After every model update the sufficient statistics of the dictionary are computed and remembered for the next model update.

The relevance prior $P(\boldsymbol{\nu})$ is used to remember the relevance parameters from the previous model update just as the dictionary prior is used to remember the dictionary. As proposed in [10] an inverse gamma distribution is used:

$$\log P(\boldsymbol{\nu}) \stackrel{c}{=} \sum_{k=1}^{K} \left[ a_k \log \nu_k + \frac{b_k}{\nu_k} \right] \tag{10}$$

$\boldsymbol{a}$ and $\boldsymbol{b}$ are the shape and scale parameters of the inverse gamma distribution and the sufficient statistics of the relevance parameters. Just like for the dictionary, the sufficient statistics of the relevance parameters are computed after every model update and remembered for the next one.

### 3.2. Inference algorithm

The inference algorithm is based on Expectation Maximisation. Every model update consists of 2 steps. In the M-step, the objective function is iteratively maximised. Just as Lee an Seung, we use multiplicative update rules because they retain the non-negativity [6]. By deriving the objective function we can find the update rules:

$$W_{fk} \leftarrow W_{fk} \sum_{n=1}^{N} \frac{V_{fn}}{(WH)_{fn}} H_{kn} + \alpha_{fk} \tag{11}$$

$$H_{kn} \leftarrow H_{kn} \frac{\sum_{f=1}^{F} \frac{V_{fn}}{(WH)_{fn}} W_{fk}}{\sum_{f=1}^{F} W_{fk} + \frac{1}{\nu_k}} \tag{12}$$

$$\nu_k \leftarrow \frac{\sum_{n=1}^{N} H_{kn} + b_k}{N + a_k} \tag{13}$$

after every update of the dictionary, it is renormalised.

In the E-step, the sufficient statistics of the dictionary and relevance parameters are computed. In the proposed model they can be computed as:

$$\alpha_{fk}^{(t+1)} = W_{fk} \sum_{n=1}^{N} \frac{V_{fn}}{(WH)_{fn}} H_{kn} + \alpha_{fk}^{(t)} \tag{14}$$

$$a_k^{(t+1)} = N + a_k^{(t)} \tag{15}$$

$$b_k^{(t+1)} = \sum_{n=1}^{N} H_{kn} + b_k^{(t)} \tag{16}$$

$.^{(t)}$ is the parameter used in model update $t$. Notice that the formula for the dictionary and its shape parameters are the same. The difference is that the shape parameters are not normalised while the dictionary is.

### 3.3. Forgetting

In equations (14)-(16) one can see that the sufficient statistics can only grow, because at every timestep only non-negative values are added to them. This causes the variance of the prior to decrease after every timestep, so the priors become more restrictive. This causes past examples to have the same impact as current examples, making it impossible to forget old information [11]. This problem is solved by introducing a forgetting factor for both the dictionary and the relevance. The forgetting factors range from zero to one. The priors are weakened by raising them to the power of the forgetting factor, thus weakening the past information after every model update, making it possible to forget old information. Raising the prior to a power is equivalent to scaling the sufficient statistics. Choosing the forgetting factors equal to zero corresponds to taking no past information into account (no memory). Choosing them equal to one corresponds to weighing all past information equally (no forgetting).

### 3.4. Dictionary resizing

While the user is using the system it is possible that they start using a new word or alternatively stop using a word. In the first case a word should be added to the dictionary and in the second case the word should be removed from the dictionary. However, it is unknown if a new example contains a new word or not. To allow the dictionary to grow we make the very naive assumption that every word in a new example is new. This means that a word will be added for every concept in the new command. If the word is not activated after the model update it is immediately removed. Words that are already in the dictionary could be removed if their relevance drops to 0. However, this would require the system to forget entirely that the word was once activated in the past, this can take a while. Instead a word is removed if it has not been activated in a number of subsequent model updates.

### 3.5. Comparison with prior work

The model proposed in this paper is similar to the models proposed in the papers [10] and [11]. In [10] both matrix priors are exponentially distributed and the scale parameters are shared between the activations and the dictionary. Sharing the scale parameters is a different way of dealing with the scaling ambiguity. In this paper this solution is not possible because the dictionary prior is needed to remember the dictionary of the previous model update, so instead the dictionary is normalised. Secondly, there is only one shape and scale parameter for all relevance parameters in [10] and they are chosen prior to deployment. In this paper the relevance shape

and scale parameters are used to remember the relevance parameters from the previous update.

In [11] the likelihood function is a multinomial distribution. However, this requires the approximation to sum to a constant value, forcing the normalisation of the activations. Normalising the activations would interfere with the ARD, so instead a Poisson likelihood is chosen. Secondly, a uniform prior is used for the activations because it is argued that the activations in previous batches contain no information about the activations in the current one. In this paper the algorithm is encouraged to activate words that were activated in previous batches, so the activations in the previous batches *do* contain information about the activations in the current one.

## 4. EXPERIMENTS

In this section the experimental results will be presented. The proposed system is compared to the system proposed in [11], meaning that no ARD is used.

### 4.1. Database

The database used in this paper is called `Syncom`. `Syncom` contains spoken commands from users for various electronic devices like a smartphone or a GPS. Each user recorded 3 sets of 16 commands. All sets contain the same semantic representations but the commands are given in a different way, e.g. "Skip this song" is a different command than "Play the next track" but the semantic representation is the same. In total there are 18 concepts in the 16 commands. 5 users have been recorded and every command was repeated 5 times, so a set contains 80 commands for every user. Not all commands that have the same semantics are completely different from each other. Some corresponding commands contain some keywords that are the same e.g. "Put on some jazz" and "Play some jazz".

### 4.2. Testing scenario

The goal of this paper is to make the system able to add new words to the dictionary or remove them if they are no longer used and everything has to be learned incrementally. To test the system's ability to do so we will simulate a user that changes their vocabulary while using the system. There are 2 phases in the testing scenario, a pre-change phase and a post-change phase. One or more sets (each containing 5 repetitions of 16 commands) are assigned to each phase. In the pre-change phase the system will be trained and tested on commands from the sets assigned to it, then the user will suddenly change their behaviour and in the post-change phase the system will be trained and tested on commands from different sets. After the change the system should be able to adjust its dictionary to the new set of commands. In each phase randomly selected batches of 16 commands from the sets assigned to it will be presented to the system. When a batch is presented to the system the dictionary will be updated and then tested on all unseen data in the assigned sets. For example, if 2 sets are assigned to the pre-change phase it means that a total of 160 commands which is a total of 10 batches are used for training and testing in this phase. First one batch will be presented to the system and the system will be tested on the remaining 9 batches. Then, another batch will be presented and the system is again tested on the remaining 8 batches and so on until 9 batches have been presented to the system. After testing the system on the last batch, this batch is presented to the system and the post-change phase starts.

All possible combinations of sets assigned to the phases will be tested. Every combination will be put into a group of similar combinations, every one of these groups is called a scenario:

- 1 set is assigned to the pre-change phase and a different one to the post change phase (1-1)
- 2 sets are assigned to the pre-change phase and the remaining one to the post change phase (2-1)
- 1 set is assigned to the pre-change phase and the remaining two to the post change phase (1-2)
- 3 sets are assigned to the pre-change phase. There are no more remaining sets, so there is no post change phase in this scenario (3-0)

For every scenario all possible set combinations are tested for all users and each one of these experiments is repeated 10 times. The experiments for different users are completely independent because the system is trained in a fully speaker dependent way. The average results for all these experiments will be reported. Because the data is presented to the system in a random order, it will be different for each repetition of the experiment.

### 4.3. Results

The average percentage of correct commands in function of the number of batches presented to the system for all scenario's can be found in table 1. A command is considered to be correct if all concepts (which is 1 or 2 for each command) are correctly recognised.

The accuracy goes up until the users change their behaviour. After the change the accuracy drops, which can be expected. Then the system starts adapting to the new vocabulary and the accuracy goes up again. The system with ARD performs better than the system without ARD at all times but specifically after the change, the system with ARD recuperates much faster. This is due to the fact that the system with ARD can add words to the dictionary that will model the new words. The system without ARD however has to first forget an old word and then learn the new word.

| | 3-0 | | 2-1 | | 1-2 | | 1-1 | |
|---|---|---|---|---|---|---|---|---|
| | ARD | base | ARD | base | ARD | base | ARD | base |
| 1 | 55.0 | 54.0 | 57.9 | 56.6 | 69.1 | 68.6 | 68.6 | 68.8 |
| 2 | 71.9 | 69.0 | 76.3 | 72.9 | 88.7 | 87.5 | 89.1 | 84.7 |
| 3 | 79.7 | 76.3 | 85.2 | 80.4 | 95.3 | 92.1 | 95.8 | 91.2 |
| 4 | 84.4 | 80.6 | 89.1 | 84.3 | 98.0 | 93.1 | 97.8 | 93.1 |
| 5 | 87.8 | 83.1 | 92.0 | 86.6 | 64.2 | 62.2 | 64.0 | 61.5 |
| 6 | 90.2 | 84.6 | 93.0 | 88.3 | 76.2 | 70.8 | 80.8 | 74.7 |
| 7 | 90.4 | 85.6 | 93.8 | 89.2 | 83.1 | 76.7 | 90.1 | 82.2 |
| 8 | 91.3 | 86.4 | 94.6 | 90.3 | 86.7 | 80.3 | 93.8 | 86.4 |
| 9 | 92.6 | 87.1 | 94.4 | 91.3 | 90.1 | 83.4 | 95.6 | 88.1 |
| 10 | 92.1 | 88.3 | 73.5 | 70.8 | 92.6 | 85.0 | | |
| 11 | 91.0 | 87.7 | 84.1 | 77.4 | 93.4 | 86.2 | | |
| 12 | 92.1 | 87.6 | 89.8 | 82.4 | 94.2 | 87.3 | | |
| 13 | 91.5 | 88.0 | 93.5 | 85.3 | 94.0 | 88.3 | | |
| 14 | 91.5 | 86.7 | 95.7 | 86.7 | 94.4 | 88.9 | | |

**Table 1**. Average percentage of correct commands. Every column corresponds to a testing scenario described in section 4.2. The results are shown for the proposed system (ARD) and the system without ARD (base). The leftmost column shows the number of batches of 16 commands that have already been presented to the system. The line marks the point where the users change their behaviour.

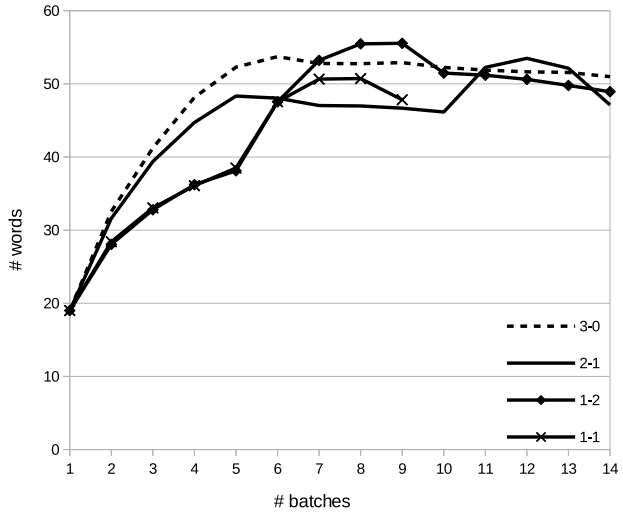


**Fig. 1**. The number of words plotted in function of the number of batches presented to the system for all scenario's described in section 4.2

In figure 1 the number of words is plotted in function of the number of presented batches for all scenarios. Only the system with ARD is shown, because the dictionary size for the system without ARD stays constant at the initial dictionary size. In the pre-change phase the model order converges to a constant value, this value is dependent on the number of command sets in the pre-change phase. The point where the change happens for the two scenarios with one set assigned to the pre-change phase is at 5 batches. At this point we can see that scenarios with more sets in the pre-change phase have a bigger dictionary, which can be expected. After the change the size jumps up because the system is detecting new words and is adding them to the dictionary. A while after the change the dictionary size starts dropping again because the words that are no longer used start getting removed. When we compare the post-change behaviour of the 1-1 scenario and the 1-2 scenario we can see that the 1-2 scenario grows to a bigger dictionary then the 1-1 scenario because there are more sets assigned to the post-change phase.

## 5. CONCLUSIONS

A method to incrementally learn the relevance of words in a dictionary has been proposed. The relevance can be used to prune the dictionary, making it possible to grow and shrink the dictionary automatically when new words have to be learned or old words forgotten. The proposed system is tested and compared to a system without ARD on scenario's where the vocabulary of the user suddenly changes. The results show that the system with ARD can adjust quicker to the new vocabulary because the dictionary can grow to model the new words.

## 6. REFERENCES


[1] Michel Vacher et al., "The sweet-home project: Audio technology in smart homes to improve well-being and reliance," in *Engineering in Medicine and Biology Society, EMBC, 2011 Annual International Conference of the IEEE*. IEEE, 2011, pp. 5291–5294.

[2] Vincent Renkens et al., "Acquisition of ordinal words using weakly supervised NMF," in *Spoken Language Technology Workshop (SLT), 2014 IEEE*. IEEE, 2014, pp. 30–35.

[3] Jort F Gemmeke et al., "A self-learning assistive vocal interface based on vocabulary learning and grammar induction," in *INTERSPEECH*, 2012.

[4] Bart Ons, Netsanet Tessema, Janneke Van De Loo, and Jort F Gemmeke, "A self learning vocal interface for speech-impaired users," *SLPAT 2013*, p. 73, 2013.

[5] Jort Gemmeke, Siddharth Sehgal, and Stuart Cunningham, "Fast vocabulary learning for disordered speech vocal interfaces," in *Spoken Language Technology Workshop (SLT)*. IEEE, 2014.

[6] Daniel D Lee and H Sebastian Seung, "Algorithms for non-negative matrix factorization," in *Advances in neural information processing systems*, 2001, pp. 556–562.

[7] Cédric Févotte, Nancy Bertin, and Jean-Louis Durrieu, "Nonnegative matrix factorization with the itakura-saito divergence: With application to music analysis," *Neural computation*, vol. 21, no. 3, pp. 793–830, 2009.

[8] Yuan Gao and George Church, "Improving molecular cancer class discovery through sparse non-negative matrix factorization," *Bioinformatics*, vol. 21, no. 21, pp. 3970–3975, 2005.

[9] David Guillamet, Bernt Schiele, and Jordi Vitria, "Analyzing non-negative matrix factorization for image classification," in *Pattern Recognition, 2002. Proceedings. 16th International Conference on*. IEEE, 2002, vol. 2, pp. 116–119.

[10] Vincent YF Tan and Cédric Févotte, "Automatic relevance determination in nonnegative matrix factorization with the beta-divergence," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 35, no. 7, pp. 1592–1605, 2013.

[11] Joris Driesen and Hugo Van hamme, "Modelling vocabulary acquisition, adaptation and generalization in infants using adaptive bayesian PLSA," *Neurocomputing*, vol. 74, no. 11, pp. 1874–1882, 2011.

[12] Geoffrey Hinton et al., "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *Signal Processing Magazine, IEEE*, vol. 29, no. 6, pp. 82–97, 2012.

[13] Hugo Van hamme, "HAC-models: a novel approach to continuous speech recognition," in *Proceedings Interspeech, ISCA*, 2008.