

Ariel University

Deep-Learning-Based Agents for Solving Novel Problems

A thesis submitted in partial fulfillment of the requirements for the degree
Doctor of Philosophy

By

Keren Nivasch

This work was prepared under the supervision of Prof. Amos Azaria



Submitted to the Senate of Ariel University

April 2023

Abstract

This work deals with several uses of deep-learning-based agents in real world settings.

The first use concerns correction detection. Intelligent agents that can interact with users using natural language are becoming increasingly common. Sometimes an intelligent agent may not correctly understand a user command or may not perform it properly. In such cases, the user might try a second time by giving the agent another, slightly different command. Giving an agent the ability to detect such user corrections might help it fix its own mistakes and avoid making them in the future.

We considered the problem of automatically detecting user corrections using deep learning. We developed a multimodal architecture that detects such user corrections. It takes as inputs the user's voice commands as well as their transcripts. Voice inputs allow the architecture to take advantage of sound cues, such as tone, speed, and word emphasis.

Another use that we considered is related to file compression. File compression is increasingly important in the internet age. Internet traffic coming from social networks, mobile apps, and the Internet of Things (IoT) cause enormous amounts of data to be stored every minute. This ever-increasing volume of data requires the usage of both physical and energy resources. In this work we used deep Reinforcement Learning (RL) methods to improve compression efficiency as well as processing time. We focused on lossless encoding techniques, in which reinforcement learning had not been applied before.

Finally, we considered the problem of keyboard layout optimization.

Since the keyboard is the most common method for text input on computers today, the design of the keyboard layout is very significant. Even though the QWERTY keyboard layout was designed more than 100 years ago, it is still the predominant layout in use today. There have been several attempts to design better layouts, both manually and automatically. We improved previous works on automatic keyboard layout optimization, by using a deep neural network to assist in a genetic search algorithm, which enables the use of a sophisticated keyboard evaluation function that would otherwise take a prohibitive amount of time. We also showed that a better choice of crossover routine greatly improves the genetic search. Finally, to test how users with different levels of experience adapt to new keyboard layouts, we conducted layout adaptation experiments with 300 participants.

Contents

Introduction	4
The papers	7
Discussion and Conclusions	35
Bibliography	36

Introduction

Deep Learning is part of a broader family of machine learning methods based on artificial neural networks. Deep learning architectures such as deep neural networks, and deep reinforcement learning, have uses in many fields including computer vision, speech recognition, natural language processing, machine translation and more.

Reinforcement Learning (RL) is a machine learning paradigm, in which an agent employs trial and error to come up with a solution to a problem, obtaining rewards or penalties for the actions it performs. The goal of the agent is to maximize the total reward. The agent starts with random trials and might finish with sophisticated tactics and skills. Deep RL based methods have recently gathered great success in several domains, such as playing Atari games, the game of Go, and self-driving cars.

Many times there is interaction between deep learning methods and other techniques, such as genetic algorithms. Genetic algorithms belong to the larger class of evolutionary algorithms. They are a technique inspired by the process of natural selection, which are commonly used to generate high-quality solutions to optimization and search problems by relying on the biologically inspired operations of mutation, crossover and selection. In a genetic algorithm there is a “population” of candidate solutions, each of which has a set of characteristics that can be altered. There is an objective function that assigns a “fitness” value to each solution. One typically starts with an initial random population, which will probably have very low fitness. The algorithm proceeds in “generations”; each generation is obtained from the previous one by selecting the most fit candidates and generating new candidates by a process of crossover. In addition, random mutations are performed on the selected candidates before being added to the next generation. While most of the crossovers and mutations are likely to reduce the fitness of the candidates, a small fraction of them will yield more-fit candidates, and the improved traits will gradually spread throughout the population. Hence, as the generations progress, the overall fitness of the population will increase.

In this work we deal with several uses of deep-learning-based agents in real world settings.

The problems considered in this work

The first use concerns correction detection. When humans interact with one another, it often happens that one person misunderstands the other. This person might then realize that she made a mistake by the other person's reaction. Therefore, she will not only correct her mistake, but she will also learn for the future what the other person's intentions were in such a situation. For personal agents to be truly useful, they should have abilities associated with human intelligence, such as the ability to detect their own mistakes from user reactions. This is an instance of *implicit feedback*, which is the gathering of information from users' behavior, as they go along normally using the agent. Implicit feedback has received a great deal of attention. It encompasses many types of user behavior: the amount of time the user spends seeing a document or a web page, her scrolling and clicking behavior, whether she copies parts of it, creates a bookmark, and so on. There have been several previous works on aspects like the correction detection problem. Levitan and Elson [1] described a method for detecting

retrieval of voice search queries. Heeman and Allen [2] considered the problem of recognizing speech repairs in spoken sentences. Paraphrase detection is closely related to our Correction Detection problem. It is the task of deciding whether two given sentences have the same meaning even though they use different words. There are several works on paraphrase detection. In particular, Kiros et al. [3] developed an off-the-shelf sentence-to-vector encoder called Skip-Thoughts, which they applied to paraphrase detection.

Another use that we considered is related to lossless file compression. Lossless data compression methods can be partitioned into two main encoding families, *statistical methods*, which include Huffman and arithmetic coding, and *dictionary methods*, in which LZ77 and LZ78 are the most famous ones. Lempel–Ziv–Welch (LZW), a practical implementation of LZ78, was developed by Welch [4] as a variant of the previous algorithm LZ78 by Abraham Lempel and Jacob Ziv. LZW employs a dictionary of strings. The dictionary is traditionally initialized by the alphabet, e.g., the set of 256 ASCII characters. The dictionary is dynamically updated as the input file is processed. Klein, Opalinsky, and Shapira [5] studied a variant of LZW in which new strings are not always added to the dictionary but only sometimes. They found that this variant has the advantage of reducing the processing time without adversely affecting the compressing ratio.

Several works applied deep learning to data compression. In most cases, these methods use deep learning strategies to predict the upcoming characters or set of characters [6]. Combining RL and data compression has only been applied on lossy compression [7,8,9]. However, to the best of our knowledge, no previous work has introduced RL to dictionary-based compression methods.

Finally, we considered the problem of keyboard layout optimization. Even though the QWERTY keyboard layout was designed more than 100 years ago, it is still the predominant layout in use today. In the early 1930's, August Dvorak introduced the keyboard layout known today as *Dvorak*, which he hoped would be more ergonomic and lead to faster typing. Dvorak has not gained much popularity, probably because QWERTY is already so entrenched. There have been several attempts to design better layouts, both manually and automatically. A popular alternative to QWERTY and Dvorak is the *Colemak* layout, introduced by Shai Coleman in 2006. Subsequently there were several attempts to find better keyboard layouts by automating the process [10,11]. Fadel et al. developed a genetic-based algorithm that is used to find better layouts than QWERTY and Dvorak. Their algorithm works by iteratively performing the operations of Selection, Crossover and Mutation, on a population of candidate layouts. Yin and Su considered several scenarios for the general keyboard arrangement problem, such as single-character and multi-character keyboards, single-finger and multi-finger typing, and optimization according to different criteria, such as typing ergonomics, word disambiguation, and prediction effectiveness. They offered an evolutionary approach using a cyber swarm method and showed that it produces keyboard layouts that are better than existing ones. Other works that use genetic algorithms for keyboard optimization are [12,13,14].

Our contributions

Regarding the correction detection problem, we developed a multimodal architecture that detects user corrections. It takes as inputs the user's voice commands as well as their transcripts. Voice inputs allow the architecture to take advantage of sound cues, such as tone, speed, and word emphasis. We also release a unique dataset in which users interacted with an intelligent agent assistant, by giving it commands. This dataset includes labels on pairs of consecutive commands, which indicate whether the latter command is in fact a correction of the former command.

Regarding file compression, we present a reinforcement learning based agent, *RLZW*, that decides when to add a string to the LZW dictionary. The agent is first trained on a large set of data, and then tested on files it has not seen previously (i.e., the test set). We show that on some types of input data, *RLZW* outperforms the compression ratio of a standard LZW.

Finally, regarding the problem of keyboard layout optimization, we improved on the previous works by using a deep neural network to assist in a genetic search algorithm, which enables the use of a more sophisticated keyboard evaluation function than was used in previous work. We also showed that a better choice of crossover routine greatly improves the genetic search. Finally, to test how users with different levels of experience adapt to new keyboard layouts, we conducted layout adaptation experiments with 300 participants.

The papers

1. Amos Azaria and Keren Nivasch, SAIF: A Correction-Detection Deep-Learning Architecture for Personal Assistants, *Sensors* 20(19):5577 (2020).
2. Keren Nivasch, Dana Shapira and Amos Azaria, Deep Reinforcement Learning for a Dictionary Based Compression Schema, *Thirty-Fifth AAAI Conference on Artificial Intelligence, AAAI 2021*: 15857-15858.
3. Keren Nivasch and Amos Azaria, Keyboard Layout Optimization and Adaptation, to appear in *IJAIT*.

Communication

SAIF: A Correction-Detection Deep-Learning Architecture for Personal Assistants

Amos Azaria ^{†,‡} and Keren Nivasch ^{*,†,‡}

Data Science Center, Ariel University, Ariel 40700, Israel; amos.azaria@ariel.ac.il

* Correspondence: kerenni@ariel.ac.il

† This paper is an extended version of our paper published in AAMAS 2019.

‡ These authors contributed equally to this work.

Received: 6 August 2020; Accepted: 23 September 2020; Published: 29 September 2020



Abstract: Intelligent agents that can interact with users using natural language are becoming increasingly common. Sometimes an intelligent agent may not correctly understand a user command or may not perform it properly. In such cases, the user might try a second time by giving the agent another, slightly different command. Giving an agent the ability to detect such user corrections might help it fix its own mistakes and avoid making them in the future. In this work, we consider the problem of automatically detecting user corrections using deep learning. We develop a multimodal architecture called SAIF, which detects such user corrections, taking as inputs the user's voice commands as well as their transcripts. Voice inputs allow SAIF to take advantage of sound cues, such as tone, speed, and word emphasis. In addition to sound cues, our model uses transcripts to determine whether a command is a correction to the previous command. Our model also obtains internal input from the agent, indicating whether the previous command was executed successfully or not. Finally, we release a unique dataset in which users interacted with an intelligent agent assistant, by giving it commands. This dataset includes labels on pairs of consecutive commands, which indicate whether the latter command is in fact a correction of the former command. We show that SAIF outperforms current state-of-the-art methods on this dataset.

Keywords: human–agent interaction; correction detection; deep learning; implicit feedback; multimodal architecture

1. Introduction

Intelligent agents that can interact with users using natural language are becoming increasingly common. Popular operating systems now come with built-in virtual assistants, such as Siri for Apple's MacOS and iOS, and Cortana for Microsoft's Windows. As another example, Amazon's Echo speakers include the Alexa virtual assistant. However, these assistants do not learn from their own mistakes, in contrast to real human assistants.

When humans interact with one another, it often happens that one person misunderstands the other. This person might then realize that she made a mistake by the other person's reaction. Consequently, she will not only correct her mistake, but she will also learn for the future what the other person's intentions were in such a situation. For example, when a manager tells her human assistant "I would like to promote Mary", the assistant might reply "Sure. I sent an email to Mary with the subject 'You're promoted'." Then the manager might reply "I would like to set a meeting to promote her". The human assistant will then probably recall the email and schedule a meeting with Mary for the promotion. The next time the manager tells the assistant she would like to promote someone, the assistant will remember to set up a promotion meeting.

For personal agents to be truly useful, they should have abilities associated with human intelligence, such as the ability to detect their own mistakes from user reactions. This is an instance of implicit feedback, which is the gathering of information from users' behavior, as they go along normally using the agent.

A personal agent with the ability to detect user corrections might be able to fix some of the mistakes it makes. For example, suppose a user says "create an email for Tom", and the agent creates a new email and sets the address to Tom's address. Then the user says "create an email and set the subject to 'for Tom'." The agent might erase the email it created and create a new email in which the subject is set to "For Tom".

In addition, an agent might learn for the future what a particular user means when giving a certain kind of request. In the above example, if later on the user says "create an email for Nancy", the agent will create a new email and set the subject to "For Nancy".

In this paper, we address the problem of detecting an agent's mistakes by identifying when the user tries to correct the agent. We refer to this problem as the *Correction-Detection* task. We develop an architecture that can detect whether given interactions constitute corrections on the part of the user or not. More precisely, the architecture works on pairs of consecutive commands. We call our architecture *Socially Aware personal assistant Implicit Feedback correction detector* (SAIF). It sees only the user's commands, and not the agent's responses to those commands, as we would like the architecture to be independent of the agent to which it is applied: A pre-trained version of the architecture should be applicable to any social agent, even though different agents have different responses.

Each pair of consecutive commands can have one of three possible labels: "new command" if the user was satisfied with the agent's action to the previous command and issued a new command; "command correction" if the user was not satisfied with the agent's action and tried to correct it; and "ASR correction" if the first command was not carried out properly due to wrong transcription by the Automatic Speech Recognition (ASR) system (for example, "set subject to Johnny" instead of "set subject to join me").

It is important to separate command corrections from ASR corrections since the actions to be taken by the agent are very different. With an ASR correction, the agent should adjust the ASR component and improve it, so that it does not fail next time. However, when dealing with a command correction, the agent should undo the previous command, and execute the learning process, as it has implicitly learned another way to say the second command.

Our architecture is multimodal, using both the voice (acoustics and non-verbal sounds) as well as the transcript of the user's spoken commands. This multimodal approach is important, since the voice input can hold important cues such as tone, speed, or emphasis on certain words. Furthermore, voice input can be especially useful in cases where the wrong command was executed due to a fault in the ASR.

Related Work

Implicit feedback has received a great deal of attention. It encompasses many types of user behavior: the amount of time the user spends seeing a document or a web page, her scrolling and clicking behavior, whether she copies parts of it, creates a bookmark, and so on. Oard and Kim [1] developed an early classification system for types of implicit feedback, based on the type of behavior, as well as based on its scope, which could be part of a document, a whole document, or a whole class of documents. Kelly and Teevan [2] later expanded this classification system. Their paper gives a broad survey of previous work on implicit feedback. Recently, Jannach et al. [3] further updated and expanded this classification system, and gave an updated survey of this area.

Search engines can use implicit feedback, such as clicking behavior, follow-up search queries and even eye-tracking, to improve the ranking of search results. The act of down-ranking one search result and up-ranking another can be considered a correction performed by the search engine in response to

the user's behavior. Implicit Feedback in search engine results often relies on the user choice among the ordered search results. Hence, it differs from the task in this work.

Levitan and Elson [4] described a method for detecting retries of voice search queries. Their task is quite similar to the one in this work, as their recognizer takes as input pairs of consecutive search commands to be classified. However, their recognizer takes as input only the transcripts of the commands. More significantly, their classification system is different, since it is binary and furthermore, if the ASR transcribed correctly, the instance is labeled as "no error", even if the user subsequently tried to correct the agent.

Zweig [5] proposed some methods for improving the accuracy of ASR translation when the user repeats her search command. In his work, recognition of repetitions is based on the fact that the user did not choose any of the options that were shown to him after his first search command. In contrast, we try to recognize user corrections from the commands themselves. Furthermore, sometimes a correction may not look like a repetition of the previous command.

Heeman and Allen [6] considered the problem of recognizing speech repairs in spoken sentences, which occur when the speaker goes back and changes or repeats something she just said. However, in our case we try to recognize when a complete command is a correction of a previous complete command.

Bechet and Favre [7] aimed to detect errors in ASR output using a combination of ASR confidence scores, and lexical and syntactic features. If the system detects a problem, it requests the user for a clarification. Ogawa and Hori [8] also aimed to detect ASR errors, using deep bidirectional RNNs. In our work, the objective is broader, since we want to detect not only ASR errors, but also user corrections unrelated to the ASR.

Paraphrase detection is the task of deciding whether two given sentences have the same meaning even though they use different words. The Microsoft Research Paraphrase Corpus [9] is a database of labeled pairs of sentences, some of which are paraphrases of one another. There are several works on paraphrase detection based on this corpus.

In particular, Kiros et al. [10] developed an off-the-shelf sentence-to-vector encoder called Skip-Thoughts, which they applied to paraphrase detection, as well as to several other learning tasks. Skip-Thoughts tries to reconstruct the surrounding sentences of an encoded passage, using the continuity of the training text. Sentences that share semantic and syntactic properties are thus mapped to similar vector representations. Skip-Thoughts also includes a vocabulary expansion method to encode words that were not seen as part of training.

Agarwal et al. [11] developed a paraphrase detection method that works well with short noisy data such as Twitter texts. See also [10,12–17].

Paraphrase detection is closely related to our Correction-Detection problem. Indeed, a user might try to correct an agent by repeating the previous command in slightly different words. For example, the user might give the command "remove the contact Tom" and the agent might not understand or not perform it correctly. The user might try again in different words by saying "delete the contact named Tom".

However, there are several differences between paraphrase detection and the Correction-Detection task. The second command might constitute a correction of the first, even though it has a slightly different meaning: The two commands might differ in proper names (e.g., Tom vs. John) or in numerical quantities, and the user's tone of voice might indicate that he got confused in the first command. Furthermore, in our task the order of the commands might be significant. For example, the agent might understand the word "create" but not the word "compose". Hence, the order between the commands "create an email for Tom" and "compose an email for Tom" is very significant.

Another similar task is the Quora Question Pairs competition, which challenges participants to tackle the problem of identifying duplicate questions [18]. Choudhary addressed this problem using BERT [19] (See also [20,21]).

Multimodal deep learning has been applied to tasks such as speech recognition, speech synthesis, emotion and affect detection, media description, and multimedia retrieval [22–27]. To the best of our knowledge this is the first research on multimodal voice and transcript deep learning for Correction Detection.

2. Materials and Methods

2.1. Formal Problem Definition

Assume a dataset of size n coming from multiple users interacting with a personal assistant agent. Let $C = \{c_1, c_2, \dots, c_n\}$ be a set of commands given to a personal agent. Each of the commands, c , is composed of a transcript of the command, c^t , the command voice, c^v , and an indicator of the agent's success in executing the command, c^s . Let $t(c_i, c_j)$ be a function that associates commands c_i and c_j with a type in $\{new, asr, cc\}$, where *new* denotes no relation between two commands (that is, the c_j is a new command), *asr* denotes that c_j was given in order to correct a malperformance of the transcription performed by the agent, and *cc* denotes that c_j is an attempt of the user to refine and correct c_i .

In this paper, we focus on the consecutive multimodal correction-detection problem, in which for each command c_i , the value of $t(c_i, c_{i+1})$ must be determined.

2.2. Dataset Description

To develop our architecture, we use a set of real interactions that users had while experimenting with the social agent *LIA* (*Learning by Instruction Agent*) [28,29]. This dataset contains a series of 2540 pairs of spoken commands given to *LIA* by 20 different users, of which 11 were male and 9 were female. The users' ages ranged from 18 to 62, with a mean of 36.9. For each command we have the original voice file and the written transcript produced by the ASR. The average command consists of 3.6 words and it lasts 4.2 seconds. Each command is followed by a response from the agent.

We manually labeled each pair of consecutive commands according to whether the second one is a correction of the first. As we mentioned, there are three possible labels: no correction ("new command"), a correction in which the user provides a different command ("command correction"), and correction due to incorrect ASR transcription ("ASR correction"). At first we tried to have the labeling done through the Mechanical Turk. However, we got very poor results, so we had to perform the labeling ourselves. Out of the 2540 commands, 568 commands are labeled as a "command correction", 236 of them are labeled as an "ASR correction", and the rest are as "new command". See Table 1 for some examples.

Table 1. Dataset examples.

Command 1	Agent Response	Command 2	Label
Create a new email	Composing new email. "outgoing email" fields are: subject, body and recipient list. What shall I do next (when executing: "tell Tom to buy milk")?	set the recipient to Tom's email address	"new command"
Create a contact	Concept "contact" was defined successfully. Please add fields to it.	a contact has an email address	"new command"
I'm running late	Sorry, but I don't understand. Would you like to teach me (say yes or just ignore)?	tell Tom I'm running late	"command correction"
How long can a whale stay underwater	Sorry, but no answer was found.	how long can whale wait underwater	"command correction"
Play the email	execute command {playYouTube}	read email	"command correction"
Who was the present of the United States	Tuesday, 20 June 2017	who was the president of the United States	"ASR correction"
In for Mariam late	Sorry, but I don't understand. Would you like to teach me (say yes or just ignore)?	inform Mary I'm late	"ASR correction"

When labeling each command, we relied on the previous command as well as on the agent's response to it to decide whether the command is a correction (even though as we mentioned, the architecture sees only the commands themselves but not the agent's responses). We also have an indicator from LIA that specifies whether the command was executed successfully or not. The dataset is available at [30].

2.3. SAIF Architecture

To address the correction-detection problem, we developed a multimodal architecture, SAIF. SAIF uses both voice and transcript inputs. Each input instance (c) consists of the voice (c^v) and transcripts (c^t) of two consecutive commands (c_i, c_{i+1}).

SAIF first converts the inputs to vector representations and encodes each command transcript (c_i^t) as a vector s_i of length 4800 using the Skip-Thoughts encoder [10] (see Section 2.4 below). SAIF then computes the component-wise product and the absolute difference of these two vectors and concatenates the results, obtaining a single vector $v_{\text{transcript}}$ of length 9600, i.e., SAIF computes $v_{\text{transcript}} = (s_i \circ s_{i+1}, |s_i - s_{i+1}|)$. To this vector, SAIF appends the feature c_i^s (marked as *exe* in Figure 1), which indicates whether the agent executed the first command or not, resulting in a vector $v'_{\text{transcript}}$ of length 9601.

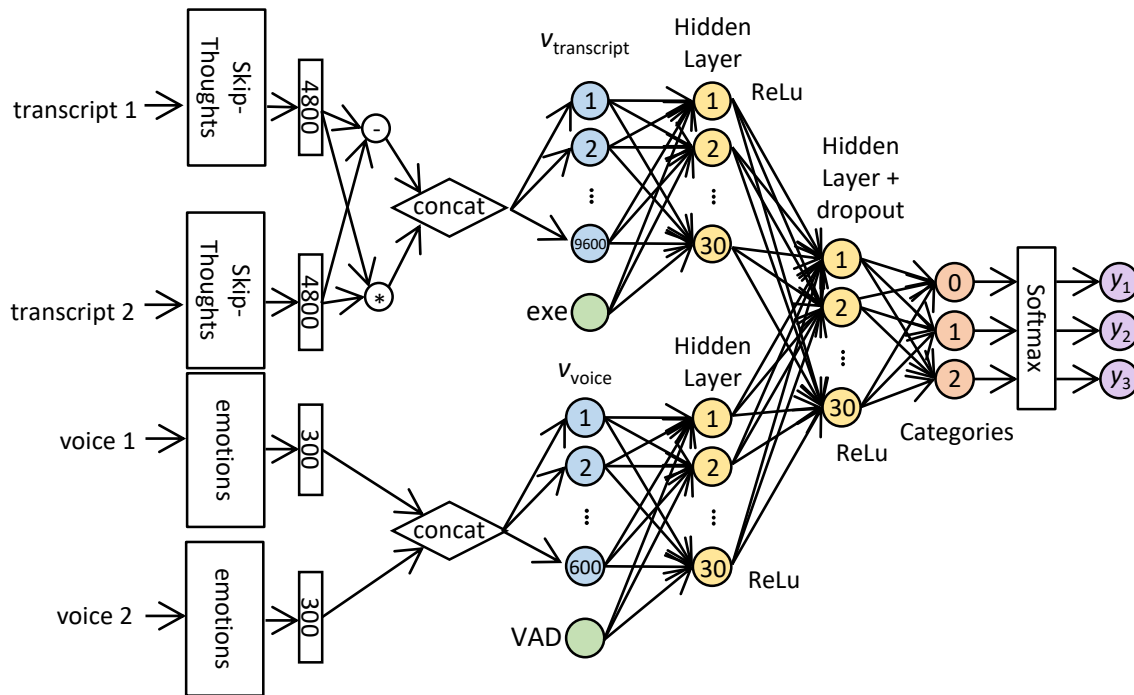


Figure 1. SAIF Architecture.

Additionally, SAIF converts the voice commands (c_i^v) into vectors. For this, it uses a model from DataFlair [23] for emotion recognition (see Section 2.4 below). Using this pre-trained model, SAIF encodes each voice file into a vector of length 300. SAIF then concatenates the encodings of the two voice commands, obtaining a vector v_{voice} of length 600. To this vector, SAIF appends a feature VAD related to voice activity detection: Using the WebRTC library [31], SAIF measures the length ℓ_i of the portion within each sound command c_i^v which constitutes actual speech. The feature VAD equals the difference $\ell_{i+1} - \ell_i$. Denote the resulting vector of length 601 by v'_{voice} .

The vector $v'_{\text{transcript}}$ is then fully connected to a Hidden Layer H_1 of 30 neurons and ReLu activation. Similarly, the vector v'_{voice} is fully connected to another Hidden Layer H_2 of 30 neurons and ReLu activation. This vector of length 60 is then fully connected to a third Hidden Layer H_3 of 30 neurons with dropout of 0.5 and ReLu activation.

The output of H_3 is linearly fully connected to a layer of size 3 which corresponds to the three possible label values. Finally, we apply SoftMax on this layer, resulting in a vector with three probabilities. The architecture is illustrated in Figure 1.

2.4. Pre-Training Methodologies

SAIF uses pre-trained models for encoding both the transcript and voice inputs. Pre-trained models enable transfer of learning and can boost accuracy without taking much time to converge, as compared to training a model from scratch.

The model used for encoding the transcripts is Skip-Thoughts by Kiros et al. [10]. This model is trained on the BookCorpus dataset which is a large collection of novels written by yet unpublished authors. The dataset has books in 16 different genres, e.g., Romance (2865 books), Fantasy (1479), Science fiction (786), Teen (430), etc. Altogether, it contains more than 74 million sentences. Along with narratives, books contain dialogue, emotion and a wide range of interaction between characters. With a large enough collection, the training set is not biased towards any particular domain or application.

Kiros et al. then expand their model's vocabulary by learning a linear mapping from a word in word2vec space to a word in the encoder's vocabulary space. The mapping is learned by using all words that are shared between vocabularies. After training, any word that appears in word2vec can then get a vector in the encoder word embedding space. Thus, even though their model was trained with only 20,000 words, after vocabulary expansion it can successfully encode almost one million possible words.

The model used for encoding the voice inputs is based on the emotion recognition model by DataFlair [23] which is pre-trained on the RAVDESS database [25] and uses a multi-layer perceptron (MLP) classifier. The RAVDESS database contains 7356 voice files from 24 actors, rated by 247 individuals 10 times on emotional validity, intensity, and genuineness. The files are labeled into eight different types of emotions (neutral, calm, happy, sad, angry, fearful, disgust, surprised). SAIF takes the last activation layer of this model to obtain a vector of size 300. The entire dataset is 24.8 GB.

3. Results

SAIF was trained and tested on the dataset mentioned in Section 2.2, as follows: An array containing all the input instances (each of which contains the voice and transcripts of two consecutive commands) was created and randomly shuffled. A 5-fold cross validation was performed: Five rounds were run, where in each round, 2032 input instances were used as training data and 508 input instances were used as test data. The training used minibatches of size 128, employing TensorFlow's Adam algorithm for optimization with a learning rate of 0.001. The training loop ran for 10000 iterations or until the train accuracy exceeded 0.995. Hence, each input instance belonged once to the test data. After averaging the results of the five tests, the obtained average test accuracy was 0.818. Since the "new command" instances constitute 68% of the data, guessing all the time "new command" would yield an accuracy of only 0.68. The SAIF code is available at [30]. Table 2 shows the confusion matrix of the results. As shown in the table, SAIF is correct most of the time.

Table 2. Confusion matrix of SAIF test results.

Actual Values	Predicted Values		
	New Command	Command Correction	ASR Correction
New command	1637	71	28
Command correction	151	378	39
ASR correction	95	78	63

In addition, Table 3 shows two groups of baselines. The first group shows some transcript-only approaches while the second group shows some voice-only approaches.

We modified SAIF to use only voice inputs or only transcripts. In these cases, the accuracy and F1 measures decreased, showing the importance of the multimodal approach. The “transcript+exe” architecture gave an accuracy slightly lower than SAIF. However, the F1 measures were noticeably lower, in particular the F1 measure of “ASR correction”.

In the first group of baselines, we show the result given by the Skip-Thoughts paraphrase detection code of [10], which was slightly modified to match our methodology. We also tried replacing Skip-Thoughts by BERT [32] in two different ways. We first tried using BERT as a text encoder, encoding each sentence separately. We also tried entering the transcript pairs in parallel following the BERT-based architecture of Choudhary [19]. In both cases, we got worse results. See Table 3.

Table 3. Comparison between different experiments.

	Accuracy	Command Correction F1	ASR Correction F1
SAIF (multimodal)	0.818	0.69	0.344
transcript+exe	0.805	0.678	0.255
transcript only	0.755	0.575	0.212
Skip-Thoughts	0.742	0.563	0.076
BERT (encoder)	0.73	0.564	0.186
BERT (2-parallel)	0.709	0.497	0.335
voice+VAD	0.68	0.03	0.047
voice only	0.677	0.006	0.015
DTW	0.681	0.012	0

In the second group of baselines, we show the result given by the Dynamic Time Warping (DTW) method [33], which measures the similarity between the two voice commands; these values then served as an input to a neural network.

We note that the Skip-Thoughts baseline method results in an accuracy of 0.742 only. Moreover, it correctly predicted only a very small number of ASR corrections. This deficiency is reflected in the very low F1 score for the “ASR correction” label. The voice-based architectures (“voice+VAD” and “voice only”) gave very poor results, and so did the DTW baseline. These three architectures guessed “new command” almost exclusively.

Clearly, SAIF achieved the best results. Among the three voice-based architectures that were tested, the “voice+VAD” slightly outperformed the other two voice-based methods, especially in detecting ASR corrections. We note that adding the voice features to the transcript features seems to help mostly in detecting ASR corrections, but also the command correction F1 slightly improves.

Discussion

As stated, the correction-detection problem is different from paraphrase detection. One difference is reflected in the fact that the order of the sentences is significant. To highlight this difference, we ran another evaluation in which we switched the order of the inputs during the test phase. This act decreased the accuracy to 0.713.

The voice component of the architecture relies on a model that is pre-trained on the RAVDESS database, which contains 7356 voice files. For comparison, the Skip-Thoughts model, which we used for the transcripts, is pre-trained on more than 74 million sentences. We believe that using a larger voice database for the pre-training will produce better voice features, which will improve the performance of the voice part of SAIF.

It might be possible to improve SAIF’s performance by making it look at three or more consecutive commands, instead of only two. For example, if the user says “set the subject to hello” and the agent responds that it does not know to which email to set the subject, then the user might try to correct the agent using two further commands: “create new mail”, and “set the subject to hello”. In cases like these, SAIF would be in a much better position if it had access to all three commands. If we refer

back to the formal definition of the correction-detection problem (See section 2.1), in the more general correction-detection problem, $t(c_i, c_j)$ must be determined for every $i < j$ and is no longer limited to $j = i + 1$ (as it is in the consecutive correction-detection problem). Furthermore, we may define $t(c_i, S)$ as a function that determines for every set (or sequence) of commands $S \subset C$ whether it is a correction of the command c_i , and, if so, what type of correction it is. It may also be possible to improve the ASR performance using the techniques of Bechet and Favre [7] and Ogawa and Hori [8], and in case of repeated utterances by using also the techniques of Zweig [5].

4. Conclusions and Future Work

In this paper, we considered the problem of automatically detecting user corrections using deep learning based on multimodal cues, i.e., text and speech. We developed a multimodal architecture (SAIF) that detects such user corrections, which takes as inputs the user's voice commands as well as their transcripts. Voice inputs allow SAIF to take advantage of sound cues, such as tone, speed, and word emphasis. We released a labeled dataset of 2540 pairs of spoken commands that users had with a social agent. The dataset includes three types of labels: "new command", "command correction", and "ASR correction". We ran SAIF on the dataset; SAIF achieved an accuracy of 0.818 and F1 measures of 0.69, 0.344 for the "command correction" and "ASR correction" labels, respectively. We showed that SAIF outperforms several other architectures, including architectures based on BERT. We believe that releasing the dataset will lead to further work on this problem.

The multimodal correction-detection problem presented in this work has many implications to social interactive agents and personal assistants. Therefore, in future work we intend to assemble SAIF in a personal agent, and use the implicit feedback obtained by correction detection to learn aliases to commands and to undo commands that were unintentionally given by the user. However, SAIF must be adjusted so that it has very high precision for the agent to be effective. High precision is required since undoing commands that the user did not intend to undo, or learning incorrect aliases, may impair the use of the agent. Assuming a high precision, the agent can learn from the examples marked as command corrections, even if the recall is relatively low. Alternatively, when suspected, the agent may explicitly ask the user whether a given command is indeed a correction, or, treat a command as a correction only if it appears as a correction more than once, or by more than a single user.

Author Contributions: Conceptualization, K.N. and A.A.; methodology, K.N. and A.A.; software, K.N.; validation, A.A.; formal analysis, K.N. and A.A.; investigation, K.N. and A.A.; resources, A.A.; data curation, K.N. and A.A.; writing—original draft preparation, K.N.; writing—review and editing, K.N. and A.A.; visualization, K.N.; supervision, A.A.; project administration, A.A.; funding acquisition, A.A. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: This work was supported by the Ministry of Science & Technology, Israel. We wish to thank the reviewers for their useful comments.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Oard, D.W.; Kim, J. Modeling information content using observable behavior. In Proceedings of the 64th Annual Meeting of the American Society for Information Science and Technology, Washington, DC, USA, 3–8 November 2001; ASIS&T: Silver Spring, MD, USA, 2001; pp. 38–45.
2. Kelly, D.; Teevan, J. Implicit feedback for inferring user preference: A bibliography. In *ACM SIGIR Forum*; ACM: New York, NY, USA 2003; Volume 37, pp. 18–28.
3. Jannach, D.; Lerche, L.; Zanker, M. Recommending Based on Implicit Feedback. In *Social Information Access-Systems and Technologies*; Springer: Berlin, Germany 2018; pp. 510–569.
4. Levitan, R.; Elson, D. Detecting retries of voice search queries. In Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), Baltimore, MD, USA, 22–27 June 2014; pp. 230–235.

5. Zweig, G. New methods for the analysis of repeated utterances. In Proceedings of the 10th Annual Conference of the International Speech Communication Association (Interspeech 2009), Brighton, UK, 6–10 September 2009; pp. 2791–2794.
6. Heeman, P.A.; Allen, J.F. Speech repairs, intonational phrases, and discourse markers: modeling speakers' utterances in spoken dialogue. *Comput. Linguist.* **1999**, *25*, 527–571.
7. Bechet, F.; Favre, B. ASR error segment localization for spoken recovery strategy. In Proceedings of the 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, Vancouver, BC, Canada, 26–31 May 2013; pp. 6837–6841.
8. Ogawa, A.; Hori, T. ASR error detection and recognition rate estimation using deep bidirectional recurrent neural networks. In Proceedings of the 2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Brisbane, Australia, 19–24 April 2015; pp. 4370–4374.
9. Dolan, B.; Quirk, C.; Brockett, C. Unsupervised construction of large paraphrase corpora: Exploiting massively parallel news sources. In Proceedings of the 20th international conference on Computational Linguistics, Association for Computational Linguistics, Geneva, Switzerland, 23–27 August 2004; p. 350.
10. Kiros, R.; Zhu, Y.; Salakhutdinov, R.R.; Zemel, R.; Urtasun, R.; Torralba, A.; Fidler, S. Skip-thought vectors. In Proceedings of the Advances in Neural Information Processing Systems, Montreal, QC, Canada, 7–12 December 2015. pp. 3294–3302.
11. Agarwal, B.; Ramampiaro, H.; Langseth, H.; Ruocco, M. A deep network model for paraphrase detection in short text messages. *Inf. Process. Manag.* **2018**, *54*, 922–937. [[CrossRef](#)]
12. Bowman, S.R.; Vilnis, L.; Vinyals, O.; Dai, A.; Jozefowicz, R.; Bengio, S. Generating Sentences from a Continuous Space. In Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning, Berlin, Germany, 11–12 August 2016; pp. 10–21.
13. Madnani, N.; Tetreault, J.; Chodorow, M. Re-examining machine translation metrics for paraphrase identification. In Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Montreal, QC, Canada, 3–8 June 2012; Association for Computational Linguistics: Stroudsburg, PA, USA, 2012; pp. 182–190.
14. Issa, F.; Damonte, M.; Cohen, S.B.; Yan, X.; Chang, Y. Abstract meaning representation for paraphrase detection. In Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers), Minneapolis, MN, USA, 2–7 June 2018; pp. 442–452.
15. Duong, P.H.; Nguyen, H.T.; Duong, H.N.; Ngo, K.; Ngo, D. A hybrid approach to paraphrase detection. In Proceedings of the 2018 5th NAFOSTED Conference on Information and Computer Science (NICS), Ho Chi Minh City, Vietnam, 23–24 November 2018; pp. 366–371.
16. El Desouki, M.I.; Gomaa, W.H.; Abdalhakim, H. A Hybrid Model for Paraphrase Detection Combines pros of Text Similarity with Deep Learning. *Int. J. Comput. Appl.* **2019**, *975*, 8887.
17. Wu, Y.; Zhang, S.; Zhang, Y.; Bengio, Y.; Salakhutdinov, R.R. On multiplicative integration with recurrent neural networks. In Proceedings of the Advances in Neural Information Processing Systems, Barcelona, Spain, 5–10 December 2016; pp. 2856–2864.
18. Quora Question Pairs | Kaggle. 2017. Available online: <https://kaggle.com/quora/question-pairs-dataset> (accessed on 27 September 2020).
19. Choudhary, D. BERT Fine-Tuning on Quora Question Pairs. 2019. Available online: https://github.com/drc10723/bert_quora_question_pairs (accessed on 27 September 2020).
20. Sharma, L.; Graesser, L.; Nangia, N.; Evci, U. Natural language understanding with the quora question pairs dataset. *arXiv* **2019**, arXiv:1907.01041.
21. Chandra, A.; Stefanus, R. Experiments on Paraphrase Identification Using Quora Question Pairs Dataset. *arXiv* **2020**, arXiv:2006.02648.
22. Baltrusaitis, T.; Ahuja, C.; Morency, L. Multimodal Machine Learning: A Survey and Taxonomy. *IEEE Trans. Pattern Anal. Mach. Intell.* **2017**, *41*, 423–443 [[CrossRef](#)]
23. Speech Emotion Recognition with Librosa. 2020. Available online: <https://data-flair.training/blogs/python-mini-project-speech-emotion-recognition/> (accessed on 27 September 2020).
24. Li, T.J.J.; Azaria, A.; Myers, B.A. SUGILITE: creating multimodal smartphone automation by demonstration. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, Denver, CO, USA, 6–11 May 2017; ACM: New York, NY, USA, 2017; pp. 6038–6049.

25. Livingstone, S.R.; Russo, F.A. The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. *PLoS ONE* **2018**, *13*, e0196391. [CrossRef] [PubMed]
26. Ngiam, J.; Khosla, A.; Kim, M.; Nam, J.; Lee, H.; Ng, A.Y. Multimodal deep learning. In Proceedings of the 28th international conference on machine learning (ICML-11), Bellevue, WA, USA, 28 June–2 July 2011; pp. 689–696.
27. Srivastava, N.; Salakhutdinov, R.R. Multimodal learning with deep boltzmann machines. In Proceedings of the Advances in Neural Information Processing Systems, Lake Tahoe, NV, USA, 3–6 December 2012; pp. 2222–2230.
28. Azaria, A.; Krishnamurthy, J.; Mitchell, T.M. Instructable Intelligent Personal Agent. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence, Phoenix, AZ, USA, 12–17 February 2016; AAAI: Phoenix, AZ, USA, 2016; pp. 2681–2689.
29. Chkroun, M.; Azaria, A. LIA: A Virtual Assistant that Can Be Taught New Commands by Speech. *Int. J. Hum. Comput. Interact.* **2019**, *35*, 1596–1607. [CrossRef]
30. Azaria, A.; Nivasch, K. Correction-Detection GitHub Repository. 2020. Available online: <https://github.com/kerenivasch/correction-detection> (accessed on 27 September 2020).
31. WebRTC. Available online: <https://webrtc.org/> (accessed on 27 September 2020).
32. Devlin, J.; Chang, M.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT, Minneapolis, MN, USA, 2–7 June 2019; Volume 1 (Long and Short Papers); Burstein, J., Doran, C., Solorio, T., Eds.; Association for Computational Linguistics: Stroudsburg, PA, USA, 2019; pp. 4171–4186.
33. Müller, M. *Information Retrieval for Music and Motion*; Springer: Berlin, Germany 2007.



© 2020 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<http://creativecommons.org/licenses/by/4.0/>).

Deep Reinforcement Learning for a Dictionary Based Compression Schema (Student Abstract)

Keren Nivasch*, Dana Shapira, Amos Azaria

Data Science Center, Ariel University, Ariel 40700, Israel
{kerenni, shapird, amos.azaria}@ariel.ac.il

Abstract

An increasingly important process of the internet age and the massive data era is file compression. One popular compression scheme, Lempel–Ziv–Welch (LZW), maintains a dictionary of previously seen strings. The dictionary is updated throughout the parsing process by adding new encountered substrings. Klein, Opalinsky and Shapira (2019) recently studied the option of selectively updating the LZW dictionary. They show that even inserting only a random subset of the strings into the dictionary does not adversely affect the compression ratio. Inspired by their approach, we propose a reinforcement learning based agent, *RLZW*, that decides when to add a string to the dictionary. The agent is first trained on a large set of data, and then tested on files it has not seen previously (i.e., the test set). We show that on some types of input data, *RLZW* outperforms the compression ratio of a standard LZW.

Introduction

Reinforcement Learning (RL) (Sutton and Barto 2017) is a machine learning paradigm, in which an agent employs trial and error to come up with a solution to a problem, obtaining rewards or penalties for the actions it performs. The goal of the agent is to maximize the total reward. The agent starts with random trials, and might finish with sophisticated tactics and skills.

Deep RL based methods have recently gathered great success in several domains, such as playing Atari games, the game of Go, and self-driving cars. However, most domains in which deep RL has been applied enjoy a fairly straightforward translation to the agent and RL domain. In this work, we apply RL techniques to the field of data compression. We propose to view both the encoder and the decoder as agents which, in different compression schemes, may be able to pick among several actions. In the context of data compression, we must use a deterministic agent so that both the encoder and the decoder take the exact same actions, and therefore are synchronized with the same world states. This is essential as the decoder must reconstruct the original uncompressed data. Therefore, in the test phase, either any element of exploration must be completely removed, or any form of

exploration must be deterministic, for example, being based upon some shared seed.

Related Work

Lossless data compression methods can be partitioned into two main encoding families, *statistical methods*, which include Huffman and arithmetic coding, and *dictionary methods*, in which LZ77 and LZ78 are the most famous ones. Lempel–Ziv–Welch (LZW), a practical implementation of LZ78, was developed by Welch (1984). LZW employs a dictionary \mathcal{D} of strings. \mathcal{D} is traditionally initialized by the alphabet, e.g. the set of 256 ASCII characters. \mathcal{D} is dynamically updated as the input file is processed by extending existing strings in \mathcal{D} by a single character, with new encountered strings that are seen in the input file for the first time. Specifically, at each stage in the compression, substrings of the input file are incrementally extended with the following character until the resulting sequence does not appear in \mathcal{D} . The code for the sequence (without the new character) is added to the output, and a new code (for the sequence concatenated to the new character) is added to \mathcal{D} . Thus, the output is a sequence of pointers to the changing dictionary. Each time the dictionary size reaches a power of 2, the number of bits used to represent the pointers increases by 1. Usually there is a bound on the dictionary size. When \mathcal{D} reaches this bound, no more strings are added to it and \mathcal{D} remains static. Alternatively, \mathcal{D} may be restarted. Klein, Opalinsky, and Shapira (2019) studied a variant of LZW in which new strings are not always added to \mathcal{D} . Rather, there exists a parameter k , and a new string is added to \mathcal{D} only every k th time. They found that this variant has the advantage of reducing the processing time without adversely affecting the compressing ratio. In this work, we develop *RLZW*, a variant of LZW, in which an RL component decides whether to insert each new string into \mathcal{D} or not. Our agent uses the Q-Learning algorithm (Sutton and Barto 2017).

While, to the best of our knowledge, no previous work has introduced RL to dictionary-based compression methods, several works applied deep learning to data compression. In most cases, these methods use deep learning strategies to predict the upcoming characters or set of characters (Shermer, Avigal, and Shapira 2010; Liu et al. 2018). Combining RL and data compression has only been applied on *lossy* compression (e.g. (Xu, Nandi, and Zhang 2003; Zhu,

*Happamon 7, Kedumim, Israel. Phone: +972-58-5405402.
Copyright © 2021, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

Lan, and van der Schaar 2013; Oladell and Huber 2012)). We note that several works have used compression in order to speed up deep learning (Ba and Caruana 2014; Amado and Meneguzzi 2018).

RLZW: Applying RL to LZW

RLZW is a neural network Q-Learning compression algorithm based on LZW. An RL agent must define states, actions and a reward function. RLZW follows the LZW algorithm, except when encountering a string that does not appear in the dictionary. Each time RLZW encounters a new string, w , RLZW is required to select between two actions: inserting it into \mathcal{D} or not. The reward function is set to the difference, in bits, between the length of the uncompressed string and the length of the corresponding pointer to the dictionary: $r = 8|w| - \lceil \log_2 |\mathcal{D}| \rceil$, where $|w|$ denotes the size of w , and $|\mathcal{D}|$ denotes the number of words in the dictionary.

The state consists of the following three parameters: A 1-hot representation of the string w , a representation of \mathcal{D} , and the number of strings that may still be added to \mathcal{D} . The representation of \mathcal{D} is composed of 0's and 1's indicating, for each possible string, whether it exists in the dictionary.

The neural network is composed of the input layer (including the encoding of the state), a hidden layer of size 30, and two output neurons (one for each action).

Experimental Settings

Due to the large size of the representation of \mathcal{D} , we used a simplified model in which the files to be compressed contain only 5 characters $\Sigma = \{-, e, g, h, t\}$. Therefore, the initialized size of \mathcal{D} is 5. We limited the maximum size for \mathcal{D} to 32 and the length of the strings in \mathcal{D} to 4; hence the number of possible strings in this model is $5^1 + 5^2 + 5^3 + 5^4 = 780$ (which is the length of the representation of \mathcal{D}).

Our dataset was composed of the *ENGLISH text collection* obtained from the *Pizza&Chili corpus*. We removed all the characters except those in Σ and created 30 files of size 18KB each. To make the compression task more challenging, we added to each file a "header" of length 50 that also contains only characters from Σ but with a different distribution than the remainder of the file. Hence, during the processing of the header, the regular LZW algorithm was expected to fill \mathcal{D} with strings that do not appear much in the remainder of the file. We hypothesised that RLZW will learn to avoid these strings.

We used 24 files for training and the remaining 6 for testing. The training was performed in 50 epochs, where in each epoch a parameter ε determined the probability of exploring (as opposed to exploiting). In the first 6 epochs ε was set to 1, and then linearly decreased until, at the last epoch, it reached 0.

Results

RLZW learned to insert into \mathcal{D} several commonly used strings (such as *the*, whereas LZW added less relevant strings. Moreover, sometimes RLZW did not fill \mathcal{D} to its full capacity, showing that it learned that with a smaller dictionary it needs fewer bits for encoding. In contrast, LZW filled \mathcal{D} quickly to its full capacity.

Algorithm	Compression Ratio
LZW	0.389
RLZW (train)	0.288
RLZW (test)	0.309

Table 1: Comparison between LZW and RLZW

Overall, RLZW succeeded to compress the training files 26% better than LZW, and the test files 21% better than LZW. See Table 1.

Conclusions and Future Work

In this paper we presented RLZW, an RL based agent that decides whether to insert the current string to the LZW dictionary or not. We showed that on some types of input data, RLZW outperformed the compression ratio of LZW.

The next steps are to extend this method to a larger alphabet and a larger dictionary size. We will consider additional reinforcement learning methods, such as a deep Actor-Critic learner. To the best of our knowledge, this work is the first to use a reinforcement learning agent in a dictionary based compression schema.

Acknowledgments

This work was supported by the Ministry of Science & Technology, Israel and by the Data Science and Artificial Intelligence Center of Ariel University.

References

- Amado, L.; and Meneguzzi, F. 2018. Q-Table compression for reinforcement learning. *Knowledge Eng. Review* 33: e22.
- Ba, J.; and Caruana, R. 2014. Do Deep Nets Really Need to be Deep? In *NIPS 2014*, 2654–2662.
- Klein, S. T.; Opalinsky, E.; and Shapira, D. 2019. Selective Dynamic Compression. In *Stringology 2019*, 102–110.
- Liu, H.; Chen, T.; Shen, Q.; Yue, T.; and Ma, Z. 2018. Deep Image Compression via End-to-End Learning. In *CVPR 2018*, 2575–2578.
- Oladell, M. C.; and Huber, M. 2012. Symbol Generation and Grounding for Reinforcement Learning Agents Using Affordances and Dictionary Compression. In *FLAIRS 2012*.
- Shermer, E.; Avigal, M.; and Shapira, D. 2010. Neural Markovian Predictive Compression: An Algorithm for On-line Lossless Data Compression. In *DCC 2010*, 209–218.
- Sutton, R. S.; and Barto, A. G. 2017. *Reinforcement Learning: An Introduction*. MIT Press, 2 edition.
- Welch, T. 1984. A Technique for High-Performance Data Compression. *IEEE Computer* 17(6): 8–19.
- Xu, W.; Nandi, A. K.; and Zhang, J. 2003. A new fuzzy reinforcement learning vector quantization algorithm for image compression. In *ICASSP 2003*, 269–272.
- Zhu, X.; Lan, C.; and van der Schaar, M. 2013. Low-complexity reinforcement learning for delay-sensitive compression in networked video stream mining. In *ICME 2013*, 1–6.

International Journal on Artificial Intelligence Tools
 © World Scientific Publishing Company

Keyboard Layout Optimization and Adaptation

Keren Nivasch

*Computer Science Department, Ariel University,
 Ariel, Israel,
 kerenni@ariel.ac.il*

Amos Azaria

*Computer Science Department, Ariel University,
 Ariel, Israel,
 amos.azaria@ariel.ac.il*

Received (Day Month Year)

Revised (Day Month Year)

Accepted (Day Month Year)

Since the keyboard is the most common method for text input on computers today, the design of the keyboard layout is very significant. Despite the fact that the QWERTY keyboard layout was designed more than 100 years ago, it is still the predominant layout in use today. There have been several attempts to design better layouts, both manually and automatically. In this paper we improve on previous works on automatic keyboard layout optimization, by using a deep neural network to assist in a genetic search algorithm, which enables the use of a sophisticated keyboard evaluation function that would otherwise take a prohibitive amount of time. We also show that a better choice of crossover routine greatly improves the genetic search. Finally, in order to test how users with different levels of experience adapt to new keyboard layouts, we conduct some layout adaptation experiments with 300 participants to examine how users adapt to new keyboard layouts.

Keywords: Keyboard Layout; Genetic Algorithm; Neural Network.

1. Introduction

The modern QWERTY keyboard layout was introduced in the 1870's by Christopher Latham Sholes¹. It has been suggested that the rationale behind the QWERTY design was to minimize type-bar jams by placing common letters far away from each other².

In the early 1930's, August Dvorak introduced the keyboard layout known today as *Dvorak*³, which he hoped would be more ergonomic and lead to faster typing. Even though the QWERTY layout is still the most common layout in use, most major operating systems offer the option of switching to the Dvorak layout. Nevertheless, Dvorak has not gained much popularity, probably because QWERTY is already so entrenched.

Despite recent advances in automatic speech recognition, keyboard input still remains the most common method of text communication used today. While people do not pay much attention to the keyboard layout, it has a tremendous impact not only on the typing speed, but also on wrist pain and repetitive strain injury (RSI)⁴. Unfortunately, the QWERTY layout so popular today was designed for typewriters rather than keyboards; therefore, it is very likely that it is sub-optimal for modern use.

There have been several other attempts at creating better keyboard layouts. A popular alternative to QWERTY and Dvorak is the *Colemak* layout⁵, introduced by Shai Coleman in 2006. It maintains the position of 17 keys of QWERTY, including many keys commonly used for keyboard shortcuts, with the hope of making it easier to learn for people accustomed to the QWERTY layout. While Colemak is not officially supported by Windows, creating and installing a custom layout in Windows can be easily done with the Microsoft Keyboard Layout Creator. Once a layout is installed, the characters appearing on the physical keys will not match the virtual characters, but this is a very minor issue; on the contrary, it encourages early adoption of touch typing.

Subsequently there were several attempts to find better keyboard layouts by automating the process^{6,7}. While a brute-force search over all possible arrangements is not feasible, due to the astronomically large number of different arrangements, there are many optimization algorithms that can be used instead. One example of a commonly used and efficient class of optimization algorithms is the genetic algorithm.

Genetic algorithms belong to the larger class of evolutionary algorithms. They are a technique inspired by the process of natural selection, which are commonly used to generate high-quality solutions to optimization and search problems by relying on the biologically inspired operations of mutation, crossover and selection. In a genetic algorithm there is a “population” of candidate solutions, each of which has a set of characteristics that can be altered. There is an objective function that assigns a “fitness” value to each solution. One typically starts with an initial random population, which will probably have very low fitness. The algorithm proceeds in “generations”; each generation is obtained from the previous one by selecting the most fit candidates and generating new candidates by a process of crossover. In addition, random mutations are performed on the selected candidates before being added to the next generation. While most of the crossovers and mutations are likely to reduce the fitness of the candidates, a small fraction of them will yield more-fit candidates, and the improved traits will gradually spread throughout the population. Hence, as the generations progress, the overall fitness of the population will increase.

In this paper we present a method for optimizing keyboard layouts using a hybrid approach of deep learning and genetic algorithms. Our method is fast and therefore allows the use of large corpora for training, as well as the use of complex fitness functions. One of the features of our method is the use of the *cycle crossover*

routine⁸, which greatly enhances the performance of the genetic part of the algorithm. We show that our method outperforms the state-of-the-art methods from the literature even when using their own metrics^a.

As mentioned in⁷, keyboard layout optimization techniques might be useful for a wider class of problems in which there are objects that must be placed in predefined locations, the objects will be accessed one after the other in some order, and the goal is to optimize the placement of the objects. Real-life examples of this scenario include books in a library and products in a vending machine.

Finally, we tackle the issue of keyboard adaptation. We examine whether experienced QWERTY typists adapt better to new layouts than inexperienced typists, whether, once experimenting with a new layout, users find it easier to adapt to another new layout, and whether it is easier or harder to adapt to common letter combinations compared to rare letter combinations. For that end, we run an experiment with 300 participants and three different keyboard layouts, namely, the standard QWERTY layout and two new layouts for three different keys.

To summarize, our contribution in the area of automatic layout optimization is three-fold.

- (1) We propose the use of deep learning to assist in a genetic algorithm process for finding an improved keyboard layout.
- (2) We show that the cycle crossover routine significantly outperforms the crossover routine that was previously used in the literature.
- (3) We conduct a user study with 300 participants to examine how users adapt to new keyboard layouts.

2. Related Work

Genetic algorithms have been used for keyboard design optimization. Yin and Su⁶ considered several scenarios for the general keyboard arrangement problem, such as single-character and multi-character keyboards, single-finger and multi-finger typing, and optimization according to different criteria, such as typing ergonomics, word disambiguation, and prediction effectiveness. They offered an evolutionary approach using a cyber swarm method and showed that it produces keyboard layouts that are better than existing ones. Other works that use genetic algorithms for keyboard optimization are^{9,10,11,7}.

In particular, in their recent work, Fadel et al.⁷ developed a genetic-based algorithm that is used to find better layouts than QWERTY and Dvorak. Their algorithm works by iteratively performing the operations of Selection, Crossover and Mutation, on a population of candidate layouts. They measure the fitness of a layout using a simple objective function that sums the Euclidean distances between

^aA link for installing the keyboard layout generated by our method on Windows is available at: <https://github.com/kerenivasch/MKLOGA>

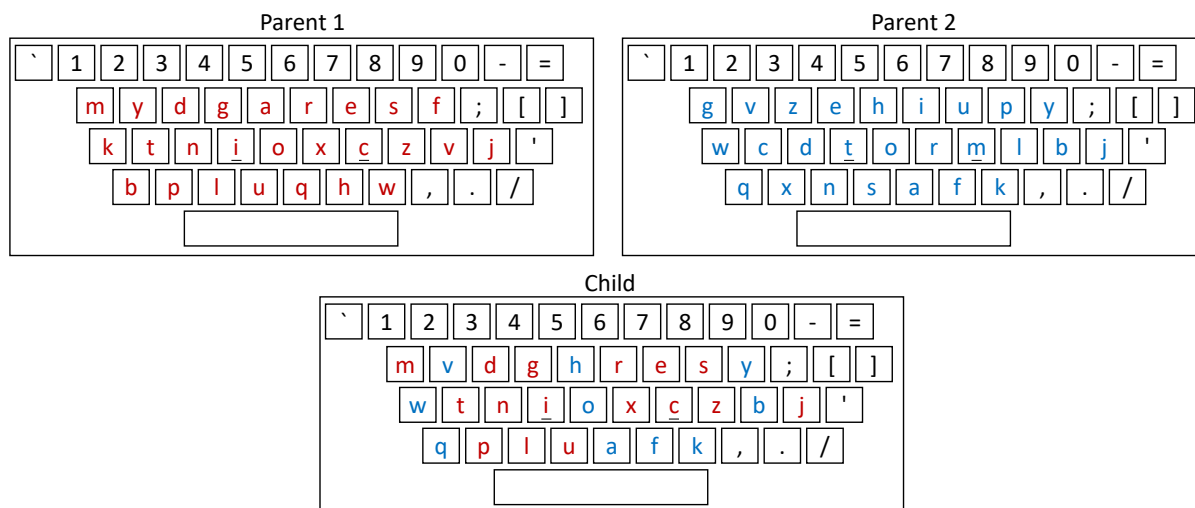


Fig. 1. Example of the cycle crossover routine.

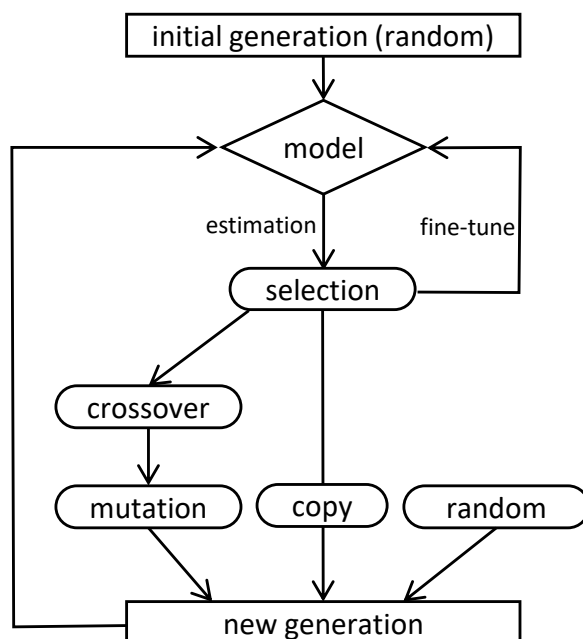


Fig. 2. MKLOGA flowchart.

every single character in the training corpus and the nearest finger to it. Using their method they find layouts that are better than QWERTY and Dvorak according to

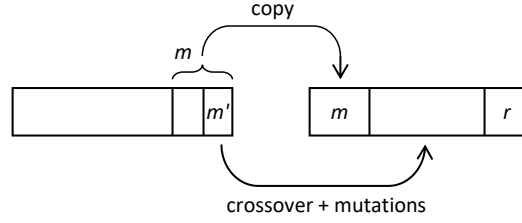


Fig. 3. The construction of the new generation from the previous one. First, r layouts are generated randomly. The best m layouts of the previous generation (according to the model estimate) are directly copied to the new generation, and the best m' of them (according to the true effort value) are used to generate new layouts through crossover and mutations.

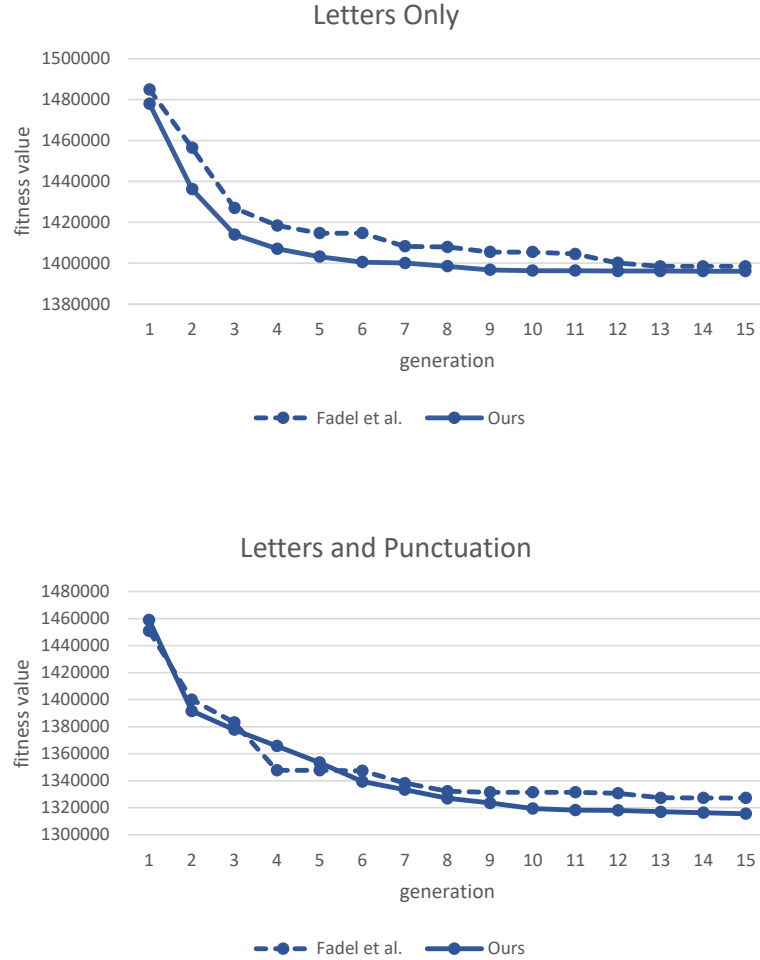


Fig. 4. A comparison between the performance of the genetic algorithm using the crossover method proposed by Fadel et al., and the cycle crossover method. The performance is measured according to the objective function defined in Fadel et al., (lower is better).

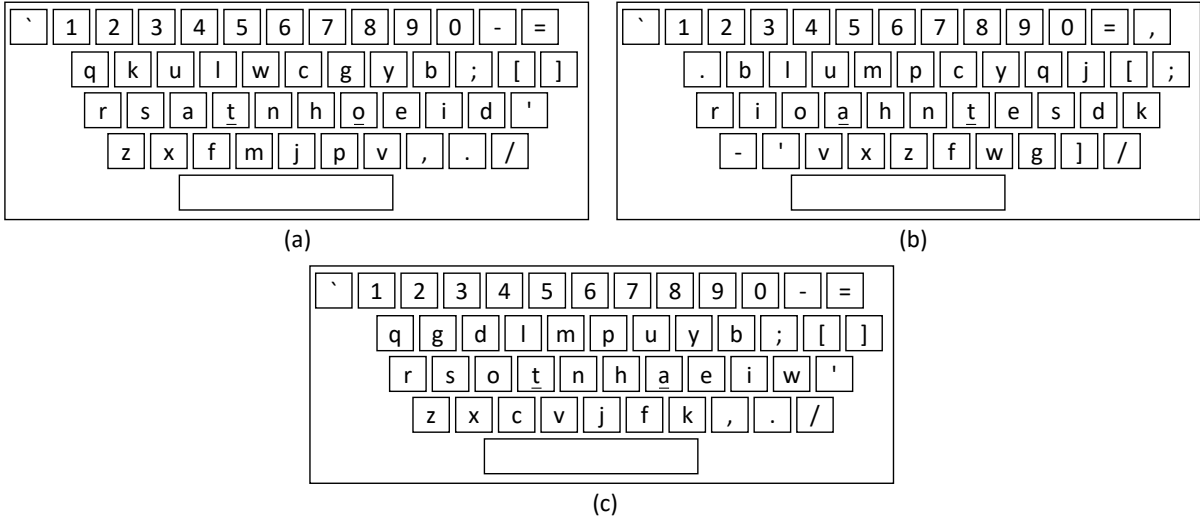


Fig. 5. (a) Best keyboard layout found by MKLOGA when moving only letter positions. (b) Best keyboard layout found by MKLOGA when moving also punctuation symbols. (c) Another keyboard layout found by MKLOGA, with ZXCVC in place.

their objective function. They call the best keyboard layout they found “}.?BZQ”.

Krzywinski¹² introduced *carpalx*, which includes a more realistic and complex objective function for evaluating layouts. The carpalx typing effort model is based on *triads*, which are three-character substrings formed from the training text. The effort associated with typing a triad has two components: effort to hit a key (independently of preceding and successive strokes) and effort to hit the group of keys. Independent effort is based on finger distance and includes hand, finger, and row penalties associated with that key. The effort associated with the group of keys considers their *stroke path*, which is evaluated by taking into account hand-alternation, row-alternation, and finger-alternation.

The carpalx model is highly parameterized, as the formula for the effort involves many weights whose value can be adjusted. Hence, the user can decide what is important to her layout, so the model can be made highly subjective. For more details on the computation of the carpalx effort model see¹². For this project we left all the carpalx parameters with their default values and did not change them.

Due to its high complexity, the carpalx objective function requires excessive computing power (approximately 0.6 seconds on a computer with Intel Core i7 CPU).

The carpalx project also includes an implementation of a simulated annealing based method for finding good keyboard layouts. Carpalx has been used to construct layouts optimized for the Filipino¹³ and Latvian languages¹⁴.

A problem related to the keyboard layout optimization problem is the Quadratic Assignment Problem (QAP). In this problem there are n facilities and n locations,

and there is a distance between each pair of facilities, as well as a flow between each pair of locations. The objective is to assign the facilities to different locations in order to minimize the sum of the distances multiplied by the corresponding flows. This problem is somewhat similar to the keyboard layout optimization problem: If the function we wish to minimize is the total movement of the fingers, then the keys and the finger base positions correspond to the facilities. There are several works that tackle the QAP problem with genetic algorithms^{15,16,17}.

There are also several previous works that combine genetic algorithms with deep learning. Sehgal et al.¹⁸ use a genetic algorithm to find the values of parameters used in a reinforcement learning task related to robotic manipulation. Potapov and Rodionov¹⁹ implement a genetic algorithm with a crossover operator that uses a deep neural network. Hu et al.²⁰ combine a genetic algorithm and deep neural network models to construct property diagrams for grain boundaries.

Recently, Klein²¹ developed a multi-step approach for generating keyboard layouts, with which he designed a new layout called *Engram*.

3. The MKLOGA Model

In this paper, we present our *Method for Keyboard Layout Optimization using a deep Genetic Algorithm (MKLOGA)*. The method improves the one described by Fadel et al.⁷ in several aspects. First, MKLOGA uses a better crossover routine for generating a new layout from its parents, as explained in section 3.1 below. In addition, MKLOGA uses the complex and more realistic objective function of carpalx¹² for evaluating layouts. Due to the excessive computing power required by the carpalx objective function, MKLOGA includes several improvements to the genetic algorithm process, one of which is the use of deep learning. All MKLOGA software is available at <https://github.com/kerenivasch/MKLOGA>.

3.1. The Cycle Crossover Routine

As mentioned above, MKLOGA uses the *cycle crossover* routine of⁸ for generating a new keyboard layout K_3 out of two given keyboard layouts K_1, K_2 . The crucial property of this routine (as opposed to the crossover routine of⁷) is that each key placement in K_3 is copied from either K_1 or K_2 . As we show in Section 5, the cycle crossover routine alone provides a significant improvement to the performance of the algorithm of⁷.

We proceed to explain the cycle crossover routine for the sake of completeness. Let S be the set of symbols whose placement is allowed to change, and let P be the set of keys that can take symbols. We first pick a random parent K from among K_1, K_2 , we call the other parent K' . Then we pick a random key $p_1 \in P$ and copy its symbol s_1 from K to K_3 . Now we check in which key p_2 , the symbol s_1 is located in K' . We copy the symbol s_2 of the key p_2 in K . Then we check in which key p_3 , the symbol s_2 is located in K' . We continue this way until we return to p_1 and thus close a cycle. Hence, all the symbols in this cycle were copied from K .

If additional keys are left, we make another random choice for K, K' between K_1, K_2 , and pick another random available key and repeat the process. This way every placement in K_3 has been copied from either K_1 or K_2 .

Figure 1 shows an example of the cycle crossover routine. Here, the routine first picked parent 1, picked from it the letter **n**, and copied the letter **n** to the child. The routine checked the location of the letter **n** in parent 2; in that location, parent 1 has the letter **l**. The routine copied the letter **l** to the child, and checked its location in parent 2. In that location, parent 1 has the letter **z**. Continuing this way, the routine copied the letter **z** and then the letter **d** to the child, and then came back to the letter **n**, which was the initial letter copied from parent 1. This finished one cycle of the crossover routine. The routine then picked parent 2, and picked from it the letter **b**. Continuing as described before, the routine copied from parent 2 to the child the letters **b, q, a, h, f, y**, and **v**, and then came back to **b** and closed another cycle. This process continued until the child layout was complete.

3.2. Using The Carpalx Objective Function

As mentioned, in order to obtain an improved keyboard layout, MKLOGA uses the complex and more realistic keyboard effort model of carpalx¹² for evaluating layouts. Since the keyboard effort model requires excessive computing power to evaluate, MKLOGA also includes a neural network for fast estimation of the effort. The neural network is initially trained on randomly generated layouts. After the training, the model is saved in order to be used as the initial model for the genetic part of the algorithm. During the genetic algorithm process, the neural network is fine-tuned by retraining it with some of the best layouts found in the current generation using their true effort value. The input layouts for the neural network are represented using a one-hot representation, as a square 0/1-matrix whose size corresponds to the number of key positions that are allowed to change. The use of the neural network allows us to evaluate the expensive effort function only on a small number of layouts, leading to a significant speedup of the running time.

The genetic algorithm of MKLOGA proceeds as a sequence of generations. Each generation consists of a population of n layouts. The first generation is generated randomly. In each generation the layouts are evaluated and sorted according to the neural network's estimation. In order to construct the new generation, r layouts are first generated randomly. The best m layouts of the previous generation pass automatically to the new generation, and they are also evaluated according to the true effort function. The best m' of these are used to generate $n - r - m$ new layouts using the cycle crossover routine. Each new generated layouts also undergoes a random number between 0 and t of random mutations. Each mutation consists of selecting 2 random keys and swapping them. Figure 2 shows a flowchart of the MKLOGA algorithm, and Figure 3 shows how a generation is constructed from the previous one.

4. Experiments

We first evaluated the effect of using the cycle crossover routine. For this, we took the code of⁷, and replaced their crossover routine with the one described in Section 3.1. We carried out the two types of experiments that were made by⁷: changing only the positions of the letters of the standard layout (called “Letters Only” in⁷), and changing also the positions of the punctuation symbols (“Letters and Punctuation”).

We then proceeded to implement MKLOGA. The first step of the implementation was to train a neural network on a data-set of 4800 randomly generated layouts labeled with their effort values. The neural network had a hidden layer of size 64 with ReLU activation. For the genetic part, we used a population size of $n = 5000$, the number of random layouts added in each generation was $r = 1000$, the parameter m was 250, and m' was 100. The maximum number of mutations was $t = 5$. We ran the genetic algorithm for 30 generations.

As a first step, we allowed to change only the positions of the letters, except for one difference: Following the lead of some previous keyboard designs (carpalx¹², colemak⁵) we moved one letter from the top row of letters to the middle row, so that the top row contains 9 letters and the middle row contains 10 letters. As a second step, we also allowed to change the positions of the punctuation symbols as in⁷.

The effort value was calculated using a corpus provided by¹² of size 267KB.

Table 1. Results with the cycle crossover

	Letters Only		Letters & Punct.	
	effort	gens	effort	gens
Fadel et al.	1394663.75	88	1312948.02	97
cycle crossover	1394663.75	16	1311932.84	27

5. Results

5.1. The Cycle Crossover Routine

The cycle crossover routine led to a significant improvement in the performance of the genetic algorithm. As depicted in Figure 4, with the cycle crossover the objective function decreased much faster. Furthermore, for the case of Letters Only, with the old crossover routine, the genetic algorithm reached the final objective function value at generation 88, whereas with the cycle crossover routine this same value was achieved at generation 16. For the case of Letters and Punctuation, the cycle crossover yielded a lower final objective function value and it was also achieved in a much earlier generation. See Table 4.

5.2. MKLOGA

As mentioned above, MKLOGA fine-tunes the neural network model during the course of the genetic algorithm. In our experiments, the loss of the model decreased

from 0.048 to 0.0028 when moving only the letter positions, and from 0.076 to 0.00098 when moving also the punctuation symbols. Hence, the neural network model’s prediction accuracy improved during the course of the execution.

When we ran MKLOGA moving only the letter positions, MKLOGA found a layout with an effort value of 1.625, and it did so already at generation 19. See Figure 5 (a). When we let MKLOGA move also the punctuation symbols, it found a layout with an effort value of 1.612, at generation 42. See Figure 5 (b). For comparison, the layout found by Fadel et al.⁷ has an effort value of 2.508 (though, as mentioned above, they optimized for a different objective function). We also ran the simulated annealing code of¹² using its default parameters. We did so 10 times and took the average effort value of the produced layouts. See Table 5.2, which also shows the effort value of a few other well known layouts, for comparison. Moreover, the neural network model of MKLOGA takes approximately only one millisecond to estimate the effort value of a layout, which is much faster than calculating the true effort.

Table 2. The carpalx effort value of different keyboards.

keyboard	effort
Qwerty	2.962
Dvorak	2.046
Colemak	1.796
Carpalx Sim. Ann.	2.038
.?BZQ (Fadel et al.)	2.508
MKLOGA Letters Only	1.625
MKLOGA Letters & Punct.	1.612

6. New Layout Adaptation Experiments

In this section we study the ability of humans to adapt to new layouts. For that end we carry out an experiment conducted with humans using multiple keyboard layouts.

6.1. *Working Hypotheses*

Our primary goal of the new layout adaptation experiments is to test the following hypotheses:

- (1) Do experienced QWERTY typists adapt better to new layouts than inexperienced typists?
- (2) Once experimenting with a new layout, would the users find it easier to adapt to another new layout?
- (3) Do users find it easier to adapt to common letter combinations than to rare letter combinations, when presented with a new layout? On the one hand, the user might recognize more quickly common letter combinations, and will also

learn more quickly to type them. On the other hand, perhaps the old ingrained habits might cause the user to get confused.

6.2. Experimental Design

We conduct an experiment in which users are requested to type certain letter combinations using different keyboard layouts. In the first phase of the experiment, the users are presented a random text of length 60 that is composed of letters and spaces. The users are requested to type this text using the standard QWERTY layout. The experiment then continues with two additional phases (phases *two* and *three*), each focusing on a different partial keyboard layout.

One partial layout consists of the letters *A, E, R* located in the positions of *D, K, S* of the QWERTY layout. The second partial layout consists of the letters *J, Z, Q* located in these same positions. For each partial layout, the users are given two typing assignments. The first assignment is only used as a warm-up and contains a random collection of the three letters and spaces, of length 60. The second assignment, for which we record results, consists of 20 triplets, separated by spaces. For the *AER* partial layout, the triplets are *ear*, *are*, and *era*, which are actual English words. For the *JZQ* partial layout, the triplets are *zjq*, *jqz*, and *zqj*, which use the same keys in the same order. One of the partial layouts is used for phase two and the other for phase three, but the order between them is chosen at random for each user. Figure 6 depicts the different phases and assignments of the experiment. In all the phases, the user cannot proceed to type the next letter until she typed correctly the current letter.

Each user is asked a few questions about themselves (age, gender, education, and previous experience with touch typing).

We use the Mechanical Turk to run this experiment with 300 different people. A software bug caused four of the users' data records to become invalid. Therefore, 296 data records are used in our analysis.

6.3. Experimental Results

As expected, users made significantly fewer mistakes with the QWERTY layout (average 6.4) than with the two new layouts (average 54.5) ($p < 0.0001$).

Quite surprisingly, we found a very small correlation between the number of errors in QWERTY and the number of errors in the two new layouts (correlation coefficients of 0.08 and 0.11 for *AER* and *JZQ* respectively). In other words, users that were good at QWERTY were not necessarily good in other new layouts. On the other hand, we found a very strong correlation between the number of errors in the two new layouts (correlation coefficient of 0.89). Hence, users that were good at one new layout were also good at the other one. We also note that learning the second new layout seems faster than learning the first new layout (regardless of which layout is being learned). Indeed, users completed the second phase significantly faster than the first (80.3s for the first and 63.2s for the second). This difference is

statistically significant ($p < 0.05$ using one-tail t-test). Interestingly however, there was no statistically significant difference between the number of mistakes in the first and second phases (53.2 and 55.7 respectively).

We were expecting a more significant difference between the *AER* and *JZQ* experiments, since in the former the triplets consisted of common characters forming actual English words, but not in the latter. However, the difference in the number of mistakes was not statistically significant (average 56.2 for *AER*, 52.7 for *JZQ*), nor was the difference in the total typing times (average 74.7s for *AER*, 68.8s for *JZQ*). A more in-depth study is required here.

Interestingly, we found no significant correlation between the users' performance (number of errors or typing time) and their age, gender, education or experience with touch typing.

Table 3. Number of mistakes made by users in each letter: QWERTY layout (top), AER layout (middle), JZQ layout (bottom).

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	space
48	66	8	22	78	32	103	49	20	131	94	71	20	46	83	71	31	31	25	41	31	134	79	49	60	88	336
a	e	r	space																							
3297	3751	3695	5908																							
j	z	q	space																							
2966	3966	3291	5401																							

Table 4. Average typing time (seconds) of each letter in the QWERTY phase (first row) compared to the Euclidean distance used by Fadel et al. (second row).

a	b	c	d	e	f	g	h	i	j	k	l	m
1.43	0.89	0.91	0.82	1.53	0.98	1.1	1.3	0.64	1.16	1.09	0.98	0.65
1.08	2.37	1.1	1.08	2.65	1.08	1.34	2.76	2.65	1.08	1.08	1.08	1.1
n	o	p	q	r	s	t	u	v	w	x	y	z
0.72	0.77	1.01	1.02	0.76	0.75	0.88	0.74	1.59	0.91	0.97	0.98	0.98
1.98	2.65	2.65	2.88	2.65	1.08	2.65	2.88	1.1	2.65	1.1	3.58	1.1

Observations on specific keys

We took a look at the number of mistakes users made on typing each letter while using the different layouts. See Table 6.3. In the QWERTY phase, users made most mistakes in keys *V* and *J* (134 and 131 respectively), and they made the fewer mistakes in keys *M* and *I* (20 for each). Interestingly, in the two other phases, users made the most mistakes in the letter located in the *K* position (letters *E* and *Z* of the new layouts), and the fewest mistakes in the letter located in the *D* position (letters *A* and *J* of the new layouts).

Since users made many mistakes when typing *J* in the QWERTY layout but they made few mistakes when typing this letter using the *JZQ* layout, this seems to indicate that the location of the letter is more significant than the letter itself.

Finally, we looked at the average time taken to type each letter during the QWERTY phase. See Table 6.3. For comparison, the table also shows the Euclidean distances used by⁷ for evaluating layout fitness. There is a correlation of -0.147 , which seems to indicate that the Euclidean distance is a poor measure of keyboard layout fitness.

7. Discussion & Future Work

As mentioned above, the best keyboard layout found by MKLOGA achieves an effort value of 1.612. In comparison, the best layout offered by the carpalx project¹² (the one they refer to as “qgmlwb”) achieves an effort value of 1.629 (in their website they give a value of 1.668, the difference being due to the use of a larger corpus). However, the layout that they recommend (which they refer to as “qgmlwy”) leaves the keys Z, X, C, V, in the classical QWERTY positions due to their frequent use in keyboard shortcuts, and it achieves an effort value of 1.635 (1.670 according to their website). Interestingly, in one run, MKLOGA produced a keyboard with the keys Z, X, C, V, in place, with an effort value of 1.633. See Figure 5 (c). Finally, it is worth noting that the above-mentioned carpalx layouts were probably built with human assistance and not purely with a computer algorithm.

MKLOGA may be useful in developing good keyboard layouts for languages other than English. Also, as mentioned in⁷, there is a need for left-hand only and right-hand only layouts for handicapped people. There is an increasing need for good layouts for smartphone keyboards, in which people type with only one finger.

In current form, MKLOGA calculates the effort value with a relatively small corpus; we plan to switch to a larger corpus in order to get more accurate results.

In addition, we plan to implement an even more realistic objective function that takes into account other relevant factors in typing. Also, we suspect that the carpalx effort model is not realistic enough, because it assigns a much better score to the Dvorak keyboard than to QWERTY, despite the fact that research shows that among experienced typewriters, it does not make much of a difference whether they use the Dvorak or the QWERTY layout. We would like a more realistic function that will explain this counter-intuitive fact so we can construct a keyboard layout that is truly better.

In the adaption experiments, we tested three different keyboard layouts. The results obtained seem to point that experienced QWERTY typists do not adapt better to new layouts, that once experimenting with a new layout users find it easier to adapt to another layout, and that the use of common letter combinations does not have a major impact on the difficulty to adapt to a new layout. However, the two new keyboard layouts were limited to three keys, since we did not want to require the participants to commit to a long-term study, and asking the participants

to learn a full keyboard layout in a short time-frame seems infeasible. In future work, we intend to perform a longitudinal study and hope to extend the results obtained in this paper to an entire keyboard layout.

8. Conclusion

Despite the fact that the QWERTY keyboard layout was designed more than 100 years ago, it is still the predominant layout in use today. There have been several attempts to design better layouts, both manually and automatically. In this paper we proposed MKLOGA, which improves on previous works on automatic keyboard layout optimization, by using a deep neural network to assist in a genetic search algorithm. As we showed, MKLOGA enables the use of a sophisticated keyboard evaluation function that would otherwise take a prohibitive amount of time. We also showed that the cycle crossover routine greatly improves the genetic search. MKLOGA produced a better keyboard layout than previous algorithms, according to the realistic typing effort model of carpalx¹². MKLOGA might be useful for developing good layouts for languages other than English, and for other situations in which objects must be placed in predefined locations.

Finally, we conducted some layout adaptation experiments with 300 participants in order to examine how users adapt to new keyboard layouts. We found that experience in QWERTY typing does not seem to make a difference in adapting better to new layouts, that once experimenting with a new layout users find it easier to adapt to another layout, and that the use of common letter combinations does not have a major impact on the difficulty to adapt to a new layout.


Acknowledgment

This research was supported in part by the Ministry of Science, Technology & Space, Israel.

References


1. J. Stamp, Fact of fiction? the legend of the QWERTY keyboard (2013), <https://www.smithsonianmag.com/arts-culture/fact-of-fiction-the-legend-of-the-qwerty-keyboard-49863249>.
2. J. Noyes, The qwerty keyboard: A review, *International Journal of Man-Machine Studies* **18**(3) (1983) 265–281.
3. N. Baker, Why do we all use qwerty keyboards? (2010), <https://www.bbc.com/news/technology-10925456>.
4. The Age, Wrist pain? try the Dvorak keyboard (2004), <https://www.theage.com.au/technology/wrist-pain-try-the-dvorak-keyboard-20041210-gdka9g.html>.
5. S. Coleman, Colemak (2006), <http://colemak.com>.
6. P.-Y. Yin and E.-P. Su, Cyber swarm optimization for general keyboard arrangement problem, *International Journal of Industrial Ergonomics* **41**(1) (2011) 43–52.
7. A. Fadel, I. Tuffaha, M. Al-Ayyoub and Y. Jararweh, Qwerty keyboard? .?BZQ is better!, in *2020 International Conference on Intelligent Data Science Technologies and Applications (IDSTA)* (Virtual, 2020), pp. 81–86.

8. I. Oliver, D. Smith and J. R. Holland, Study of permutation crossover operators on the traveling salesman problem, in *Genetic algorithms and their applications: proceedings of the second International Conference on Genetic Algorithms* (Massachusetts Institute of Technology, Cambridge, MA, 1987).
9. C. Liao and P. Choe, Chinese keyboard layout design based on polyphone disambiguation and a genetic algorithm, *International Journal of Human-Computer Interaction* **29**(6) (2013) 391–403.
10. M. Govind and V. V. Panicker, Optimization of a single finger keyboard layout using genetic algorithm and topos, *International Journal of Scientific & Engineering Research* **7**(2) (2016) 102–105.
11. A. H. H. Onsorodi and O. Korhan, Application of a genetic algorithm to the keyboard layout problem, *PLOS ONE* **15** (01 2020) 1–11.
12. M. Krzywinski, Carpalx keyboard layout optimizer (2005), <http://mkweb.bcgsc.ca/carpalx/>.
13. J. M. R. Salvo, C. J. B. Raagas, M. T. C. M. Medina and A. J. A. Portus, Ergonomic keyboard layout designed for the filipino language, in *Advances in Physical Ergonomics and Human Factors* (Springer, 2016) pp. 407–416.
14. V. Vitolins, Modernized latvian ergonomic keyboard, *arXiv preprint arXiv:1707.03753* (2017).
15. H. Azaronyad and R. Babazadeh, A genetic algorithm for solving quadratic assignment problem (qap), in *Proceeding of 5th International Conference of Iranian Operations Research Society (ICIORS)* (Tabriz, Iran, 2012).
16. R. K. Ahuja, J. B. Orlin and A. Tiwari, A greedy genetic algorithm for the quadratic assignment problem, *Computers & Operations Research* **27**(10) (2000) 917–934.
17. A. Hameed, B. Aboobaider, M. Mutar and N. Choon, A new hybrid approach based on discrete differential evolution algorithm to enhancement solutions of quadratic assignment problem, *International Journal of Industrial Engineering Computations* **11**(1) (2020) 51–72.
18. A. Sehgal, H. La, S. Louis and H. Nguyen, Deep reinforcement learning using genetic algorithm for parameter optimization, in *2019 Third IEEE International Conference on Robotic Computing (IRC)* (Naples, Italy, 2019), pp. 596–601.
19. A. Potapov and S. Rodionov, Genetic algorithms with dnn-based trainable crossover as an example of partial specialization of general search, in *International Conference on Artificial General Intelligence* Springer, (Seattle, WA, 2017), pp. 101–111.
20. C. Hu, Y. Zuo, C. Chen, S. P. Ong and J. Luo, Genetic algorithm-guided deep learning of grain boundary diagrams: addressing the challenge of five degrees of freedom, *Materials Today* **38** (2020) 49–57.
21. A. Klein, Engram: a systematic approach to optimize keyboard layouts for touch typing, with example for the English language (2021).




Let's start with a test on the QWERTY layout. Please type the following:

oym gano ca qip l jnvymjtsrbwl p nezrlawrztwwxk h rrqr
uge




Now, use the layout shown above. Let's start with a warm up:

eaae arrrrr raare aaaraar aaeearae rrareerra ee aeee a ee
are




Great! Now type the following using this layout:

are are era era are are ear are are are era are ear are a
re are ear are era era



Now, use the new layout shown above. Let's start with a warm up:

qqzqjzqz z qq zjqqqjzqq zj z qzjqzqqjzz j z z qqzjzqqj
jq



Great! Now type the following using this layout:

jqz zqj jqz zjq zjq zjq zqj zqj zjq zqj zjq zqj j
qz zjq zjq zjq jqz zqj

Fig. 6. The typing experiments

Discussion and Conclusions

In this work we considered the problem of automatically detecting user corrections using deep learning based on multimodal cues, i.e., text and speech.

We developed a multimodal architecture (SAIF) that detects such user corrections, which takes as inputs the user’s voice commands as well as their transcripts.

Voice inputs allow SAIF to take advantage of sound cues, such as tone, speed, and word emphasis. We released a labeled dataset of 2540 pairs of spoken commands that users had with a social agent. We believe that releasing the dataset will lead to further work on this problem.

The multimodal correction detection problem presented in this work has many implications to social interactive agents and personal assistants. Therefore, future work might include assembling SAIF in a personal agent, and using the implicit feedback obtained by correction detection to learn aliases to commands and to undo commands that were unintentionally given by the user.

We also presented RLZW, an RL based agent that decides whether to insert the current string to the LZW dictionary or not. We showed that on some types of input data, RLZW outperformed the compression ratio of LZW. To the best of our knowledge, this work is the first to use a reinforcement learning agent in a dictionary-based compression schema. Future work might include extending this method to a larger alphabet and a larger dictionary size as well as employing additional reinforcement learning methods, such as a deep Actor-Critic learner.

Regarding keyboard layout optimization, our solution may be useful in developing good keyboard layouts for languages other than English. Also, there is a need for left-hand only and right-hand only layouts for handicapped people. There is an increasing need for good layouts for smartphone keyboards, in which people type with only one finger.

We suspect that the typing effort model that we used is not realistic enough, because it assigns a much better score to the Dvorak keyboard than to QWERTY, even though research shows that among experienced typewriters, it does not make much of a difference whether they use the Dvorak or the QWERTY layout. We would like a more realistic function that will explain this counter-intuitive fact so we can construct a keyboard layout that is truly better.

In the typing adaption experiments, we tested three different keyboard layouts. The results obtained seem to point that experienced QWERTY typists do not adapt better to new layouts, that once experimenting with a new layout users find it easier to adapt to another layout, and that the use of common letter combinations does not have a major impact on the difficulty to adapt to a new layout. However, the two new keyboard layouts were limited to three keys, since we did not want to require the participants to commit to a long-term study, and asking the participants to learn a full keyboard layout in a short time-frame seems infeasible. Future work might include performing a longitudinal study and hope to extend the results to an entire keyboard layout.

Bibliography

1. Levitan, R.; Elson, D. Detecting retries of voice search queries. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), 2014, pp. 230–235.
2. Heeman, P.A.; Allen, J.F. Speech repairs, intonational phrases, and discourse markers: modeling speakers' utterances in spoken dialogue. *Computational Linguistics* 1999, 25, 527–571.
3. Kiros, R.; Zhu, Y.; Salakhutdinov, R.R.; Zemel, R.; Urtasun, R.; Torralba, A.; Fidler, S. Skip-thought vectors. *Advances in neural information processing systems*, 2015, pp. 3294–3302.
4. Welch, T. 1984. A Technique for High-Performance Data Compression. *IEEE Computer* 17(6): 8–19.
5. Klein, S. T.; Opalinsky, E.; and Shapira, D. 2019. Selective Dynamic Compression. In *Stringology 2019*, 102–110.
6. Shermer, E.; Avigal, M.; and Shapira, D. 2010. Neural Markovian Predictive Compression: An Algorithm for Online Lossless Data Compression. In *DCC 2010*, 209–218.
7. Xu, W.; Nandi, A. K.; and Zhang, J. 2003. A new fuzzy reinforcement learning vector quantization algorithm for image compression. In *ICASSP 2003*, 269–272.
8. Zhu, X.; Lan, C.; and van der Schaar, M. 2013. Low complexity reinforcement learning for delay-sensitive compression in networked video stream mining. In *ICME 2013*, 1–6.
9. Oladell, M. C.; and Huber, M. 2012. Symbol Generation and Grounding for Reinforcement Learning Agents Using Affordances and Dictionary Compression. In *FLAIRS 2012*.
10. P.-Y. Yin and E.-P. Su, Cyber swarm optimization for general keyboard arrangement problem, *International Journal of Industrial Ergonomics* 41(1) (2011) 43–52.
11. A. Fadel, I. Tuffaha, M. Al-Ayyoub and Y. Jararwch, Qwerty keyboard? .?BZQ is better!, in 2020 International Conference on Intelligent Data Science Technologies and Applications (IDSTA) (Virtual, 2020), pp. 81–86.
12. C. Liao and P. Choe, Chinese keyboard layout design based on polyphone disambiguation and a genetic algorithm, *International Journal of Human-Computer Interaction* 29(6) (2013) 391–403.
13. M. Govind and V. V. Panicker, Optimization of a single finger keyboard layout using genetic algorithm and topos, *International Journal of Scientific & Engineering Research* 7(2) (2016) 102–105.
14. A. H. H. Onsorodi and O. Korhan, Application of a genetic algorithm to the keyboard layout problem, *PLOS ONE* 15 (01 2020) 1–11.

תקציר

עבודה זו עוסקת במספר שימושים של סוכנים המבוססים על למידה עמוקה בבעיות מהעולם האמיתי.

השימוש הראשון נוגע לזיהוי תיקונים. סוכנים חכמים שיכולים ליצור אינטראקציה עם משתמשים באמצעות שפה טבעית הופכים נפוצים יותר ויותר. לפעמים סוכן חכם עלול לא להבין נכון פקודת משתמש או לא לבצע אותה כראוי. במקרים כאלה, המשתמש עשוי לנסות פעם שנייה על ידי מתן פקודה נוספת, מעט שונה לסוכן. מתן היכולת לסוכן לזהות תיקוני משתמשים כאלה עשוי לעזור לו לתקן את הטעויות שלו ולהימנע מלעשות אותן בעתיד. אנו שקלנו את הבעיה של זיהוי אוטומטי של תיקוני משתמשים באמצעות למידה עמוקה. פיתחנו ארכיטקטורה מולטי-מודאלית שמזהה תיקוני משתמשים כאלה. הארכיטקטורה מקבלת כקלט את הפקודות הקוליות של המשתמש כמו גם את התמלילים שלו. כניסות קוליות מאפשרות לארכיטקטורה לנצל רמזים שנמצאים בקול, כגון טון, מהירות והדגשת מילים.

שימוש נוסף שחקרנו קשור לדחיסת קבצים. דחיסת קבצים חשובה יותר ויותר בעידן האינטרנט. תעבורת אינטרנט המגיעה מרשתות חברתיות, אפליקציות סלולריות ואינטרנט של הדברים (IoT) גורמת לאחסנת כמויות אדירות של נתונים בכל דקה. נפח הנתונים ההולך וגדל הזה דורש שימוש במשאבים פיזיים ואנרגיה. בעבודה זו השתמשנו בשיטות למידת חיזוק עמוקה (RL) כדי לשפר את יעילות הדחיסה כמו גם את זמן העיבוד. התמקדנו בטכניקות קידוד ללא אובדן, שבהן למידת חיזוק לא יושמה קודם לכן.

לבסוף, שקלנו את הבעיה של אופטימיזציה של עיצוב פריסת מקלדת. מכיוון שהמקלדת היא האמצעי הנפוץ ביותר להזנת טקסט במחשבים כיום, עיצוב פריסת המקלדת הוא משמעותי ביותר. למרות שפריסת מקלדת ה-QWERTY (באנגלית) תוכננה לפני יותר מ-100 שנה, היא עדיין הפריסה השולטת בשימוש כיום. היו מספר ניסיונות לעצב פריסות טובות יותר, הן באופן ידני והן באופן אוטומטי. אנו שיפרנו עבודות קודמות של אופטימיזציה אוטומטית של פריסת מקלדת, על ידי שימוש ברשת עצבית עמוקה כדי לסייע באלגוריתם חיפוש גנטי, המאפשר שימוש בפונקציית הערכת מקלדת משוכללת שבמצב אחר היה לוקח פרק זמן עצום. הראינו גם שבחירה טובה יותר של שחלוף משפרת מאוד את החיפוש הגנטי. לבסוף, כדי לבדוק כיצד משתמשים בעלי רמות ניסיון שונות מסתגלים לפריסות מקלדת חדשות, ערכנו ניסויים להתאמת פריסה עם 300 משתתפים.

אוניברסיטת אריאל בשומרון

סוכנים מבוססי למידה עמוקה לפתרון

בעיות חדשות

חיבור זה הוגש כחלק מדרישות התואר
"דוקטור לפילוסופיה"

מאת

קרן ניבש

עבודה זו נכתבה בהנחיית פרופ' עמוס עזריה

מוגש לסנאט אוניברסיטת אריאל בשומרון

אפריל 2023