Text-line-up: Don't Worry about the Caret

Chandranath Adak[1,2], Bidyut B. Chaudhuri[3,4], Chin-Teng Lin[2], and Michael Blumenstein[2]

[1] JIS Institute of Advanced Studies & Research, JIS University, India-700091
[2] Australian AI Institute, University of Technology Sydney, Australia-2007
[3] Techno India University, India-700091
[4] CVPR Unit, Indian Statistical Institute, India-700108
chandra@jisiasr.org

Abstract. In a freestyle handwritten text-line, sometimes words are inserted using a caret symbol (^) for corrections/annotations. Such insertions create fluctuations in the reading sequence of words. In this paper, we aim to line-up the words of a text-line, so that it can assist the OCR engine. Previous text-line segmentation techniques in the literature have scarcely addressed this issue. Here, the task undertaken is formulated as a path planning problem, and a novel multi-agent hierarchical reinforcement learning-based architecture solution is proposed. As a matter of fact, no linguistic knowledge is used here. Experimentation of the proposed solution architecture has been conducted on English and Bengali offline handwriting, which yielded some interesting results.

Keywords: Handwriting · Hierarchical reinforcement learning · Multi-agent reinforcement learning · Proof-reading.

1 Introduction

Extracting the text-line information from a handwritten page image is a classical problem of document image analysis [1–4], and it is still prominent in this deep learning era [1]. Most of the past works in this direction have focused on text-line segmentation [2], where primarily the text-lines are either separated through a continuous fictitious line [3], or labeled by clusters [4], or marked by baselines [1]. A text-line may contain some words that are written later using the caret symbol (^) for correcting/annotating the manuscript. Here, a text-line segmentation approach may not work well to ascertain whether the inserted words belong to a certain text-line (refer to Fig. 1). As a consequence, it may impede the understanding of the reading sequence of words of a text-line while OCRing.

In this paper, we undertake the task to comprehend the sequence of words of a text-line, so that an OCR engine/automated manuscript-transcriptor can have some prior knowledge of the reading sequence of words. To handle this task, some character recognition followed by natural language processing (NLP) can be performed, but that would be a costly procedure. Therefore, we neither use here character recognition nor linguistic knowledge. Another way involves detecting

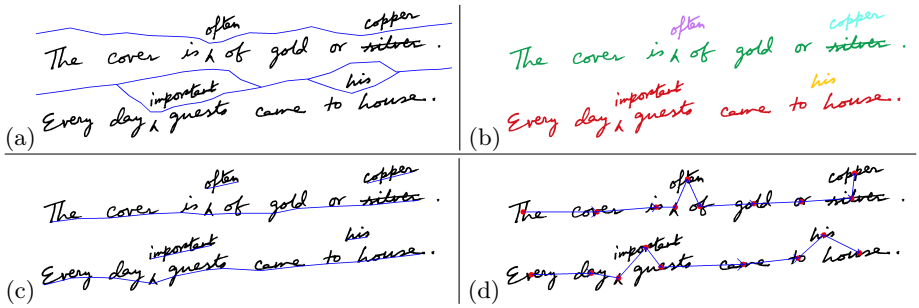

Fig. 1: Text-line representation: (a) separated by a fictitious line [3], (b) clusters [4], (c) marked by a baseline [1], (d) location coordinates and detected path (arrowed-line) of the reading sequence [ours].

the caret symbols and then attempting to analyze the surrounding areas, but this approach is not fruitful due to false-negative and false-positive cases, e.g., small-sized carets close/overlapping with texts, tiny character graphemes-like carets, word inserted without using a caret, no word insertion after scribbling a caret, etc.

We formulate the above task as a path planning problem, where the system agent attempts to visit every word component exactly once to detect the reading sequence of a text-line. Here, we propose a multi-agent hierarchical reinforcement learning model [5] to impart the machine with a human-like perception of reading a text-line that may be obstructed due to some inserted texts. The motivation to tackle this problem using reinforcement learning is its working strategy of exploring the unknown terrain while exploiting the current knowledge [6]. From the application point of view, this work is significant for analyzing the reading-word sequence of a text-line, especially for freehand writing, where the past methods did not perform well. To the best of our knowledge, our work is the earliest attempt of its kind. From the theoretical perspective, we propose a novel multi-agent hierarchical reinforcement learning architecture, where we define the relationship among agents, their interactions with the environment, and shape the global and internal reward.

We performed the experiments on English and Bengali offline handwriting, which are left-to-right writing systems. The challenges of our employed datasets concerning the undertaken task are discussed in Section 2. In Section 3, we formulate the problem and propose a solution architecture. Section 4 presents and analyzes the experimental results. Finally, Section 5 concludes this paper.

2 Challenges and Dataset Details

In this section, we discuss the offline handwriting dataset employed for our research and the challenges related to the data. The primary aim of this research is to analyze the reading sequence of words from a handwritten text-line without any assistance from OCR and NLP engines.

The freestyle handwritten text-lines are mostly curvilinear rather than linear [4] and pose several challenges for our research. The classical challenges include text-line fluctuation and orientation, e.g., skewness, waviness, curviness. The variation in the inter text-line gap draws significant attention, where the neighboring text-lines may be close, or touching, even overlapping [2]. Similar kinds of issues can be found for the intra text-line gap, i.e., gap between words/inter-word gap.

In daily handwriting and manuscript drafting/corrections, several forms of annotations can be found; therefore, detection of the word-reading sequence is not a straight-forward problem. A common form is inserting a word between two successive words by using a caret symbol (^), and writing the word in the inter text-line gap between the current and the previous text-line. Here, the insertion of multiple words can also be noted. The habitual absence of caret symbols is also possible, which leads to more challenging scenarios. The writer often strikes-through formerly written word(s) and inserts some substituting word(s). Such insertions (with/without a caret), deletion (strike-through), update (deletion + insertion) are noted frequently in real-time writing (refer to Fig. 1). The font-size of the inserted word(s) may be smaller to fit into the inter-text-line gap. If this gap is not sufficient to insert the word(s), then the writer may write it in the available marginal-space of the page. Such word insertions depend on the writing space availability. Sometimes, the forceful insertion of words almost removes the usual inter-word and inter-line gaps, which makes it harder to read. False-positive cases may also arise, where some words may be incorrectly thought of as inserted ones, due to the absence of the caret, lower inter text-line gaps, artistic/poetic writing structure, superscript-text, etc. Moreover, the presence of some unconventional annotations performed by the human-writer is quite natural, due to handwriting variations, improper/no formal training of text-annotation, idiosyncratic writing styles, etc.

For our research, we require a database, which contains some handwritten pages having multiple words inserted during text-annotation; so that we can perform our experiments on detecting the reading sequence of words. The handwritten pages of publicly available databases hardly contain text-annotations as per our requirement. Therefore, we manually annotated some handwritten page images of publicly available databases using the GIMP image editor [7]. On a page image, we carefully inserted some words after extracting/cropping those words from the same page. Here, we used 100 English pages from the IAM database [8] and 100 Bengali pages from the PBOK database [9], and called these as $\mathrm{DB_{IAM}}$ and $\mathrm{DB_{PBOK}}$, respectively. Here, 50% of the pages of both $\mathrm{DB_{IAM}}$ and $\mathrm{DB_{PBOK}}$ were annotated/inserted by linguistically meaningful words using carets. For the remaining 50% of pages, we inserted linguistically non-meaningful words with/without a caret, which did not impede our objective to learn the structural pattern and position of inserted handwritten word(s). Moreover, to address the natural flow of annotations during the free-style writing, i.e., insertion of the word(s) by the same writer, we procured 50 pages of English (say, $\mathrm{DB_E}$) and 50 pages of Bengali (say, $\mathrm{DB_B}$) offline handwriting by an in-house

Table 1: Employed dataset details

Dataset	Script	# page	r_{OTL} $(avg \pm sd)$
DB_{IAM}	English	100	0.5203 ± 0.1841
DB_{PBOK}	Bengali	100	0.4764 ± 0.1160
DB_{E}	English	50	0.3725 ± 0.2748
DB_{B}	Bengali	50	0.3551 ± 0.2377

setup. In this paper, we consider English and Bengali scripts, which are usually written horizontally from left-to-right. In Table 1, we summarize some aspects of our employed datasets. Inserting some word(s) to a text-line obstructs the straight-forward reading path/sequence. On a page, we count such obstructed text-lines (n_{OTL}) and divide it with the total number of text-lines (n_{TL}) of that page, to get a ratio, say, $r_{OTL} = n_{OTL}/n_{TL}$. The average $\pm$ standard deviation ($avg \pm sd$) of r_{OTL} over a dataset is mentioned in Table 1. For the available datasets in the literature, r_{OTL} $(avg) \approx 0$, due to their different objectives from ours.

In this paper, we attempt to imitate the human perception of detecting the word sequence while reading a text-line. Therefore, to prepare the ground-truth, we engaged human volunteers having at least professional working proficiency in the English/Bengali language. A volunteer was requested to perform computer-mouse clicks on the words in the same sequence of his/her reading from the digital image of a handwritten page projected on a computer screen, so that we can record the (x_i, y_i) coordinates of the word sequence through mouse-clicks. A very few noisy mouse clicks were manually discarded with opinions from some linguistic experts. We also extracted the CG (x_c, y_c) of a word component semi-automatically. We did not fully rely on the CG due to our objective of mimicking the eye movement during human gazing/reading. At least n_r (> 1) number of readers were engaged for a page. Subsequently, the corresponding ground-truth coordinate $(x_{GT},\ y_{GT})$ of a word is computed as follows. $(x_{GT},\ y_{GT}) = (\frac{\alpha_r}{n_r} \sum_{i=1}^{n_r} x_i + \alpha_c x_c,\ \frac{\alpha_r}{n_r} \sum_{i=1}^{n_r} y_i + \alpha_c y_c)$; where, $\alpha_r + \alpha_c = 1$. For our experimental dataset generation, we chose $n_r = 10, \alpha_r = 0.5$, and $\alpha_c = 0.5$. In our dataset, for a handwritten page, the word ground-truth coordinates are provided sequentially with the demarcation of text-lines.

3 Proposed Method

In this section, we first formulate the undertaken problem, then propose our solution architecture.

3.1 Problem Formulation

In this research work, we are given an image ($\mathcal{I}$) obtained by scanning a handwritten page, which is fed to our system as an input. Our task is to detect the

reading-sequence of words in a text-line only by the structural pattern of handwriting. No linguistic knowledge has been used here. As we mentioned earlier, the sequence of words in a text-line does not always follow a straight path due to the insertion of some words during proof-reading/corrections. The task is formulated here as a path planning problem, where the system-agent visits every word component exactly once. The source, target, and in-between hopping locations of the path are obtained by the agent itself through an exploration-exploitation strategy. The system outputs the sequence of these location coordinates over $\mathcal{I}$, which eventually infers the path.

3.2 Solution Architecture

As we stated previously, we refrain from obtaining assistance from any OCR engine followed by an NLP architecture to comprehend the reading sequence of words in the text-line due to the high cost. Instead, we think of composing our problem as a decision-making task, where an agent interacts with the handwriting image (environment) to find the sequence of word-locations. For such cases, a reinforcement learning (RL)-based agent is a good option, since it learns a policy for maximizing the reward by taking action on the environment [6]. However, a full handwritten page image is a challenging environment for an agent to interact, owing to various complex writing patterns. To alleviate the learning complexity, here, a hierarchical reinforcement learning (HRL)-based architecture can be proposed, which breaks down the task into sub-tasks hierarchically [5,10]. Moreover, at a certain level of the hierarchy, multiple agents can work together for better learning and communication among themselves [5]. Altogether, in this paper, we propose a multi-agent hierarchical reinforcement learning (MAHRL) model to achieve our goal.

Architecture Overview. For the undertaken task, our model contains two levels of hierarchy. In the higher level, one RL agent (manager) is present, while the lower level includes n_w number of RL agents (worker-1, worker-2, ..., worker-n_w) that depends on the count of text-lines on a page. In Fig. 2, we represent our MAHRL model diagrammatically. Formally, the RL problem can be built as an MDP (Markov Decision Process), which consists of a set of agent *states* of the environment (s), a set of *actions* (a) to attain the goal, and a *reward* function (r) to optimize the decision strategy [6]. The manager observes the entire environment ($\mathcal{E}$)/handwritten page to encode its state space (s^m); manager's action (a^m) refers to assigning a sub-task to a worker. The manager receives a *global reward* (r^m) from the environment after completion of the entire work. For the worker RL agent, we formulate a partially observable MDP [11], where the agent stochastically makes a decision in discrete-time without observing the entire environment [6]. The worker-i ($\forall\ i = 1, 2, \ldots, n_w$) observes the partial environment $\mathcal{E}^{w_i}$ from a handwritten page to embed its state (s^{w_i}) and acts (a^{w_i}) to find the sequence of word-locations from a text-line. The worker-i receives an *internal reward* (r^{w_i}) from the manager. The workers share weights among themselves.

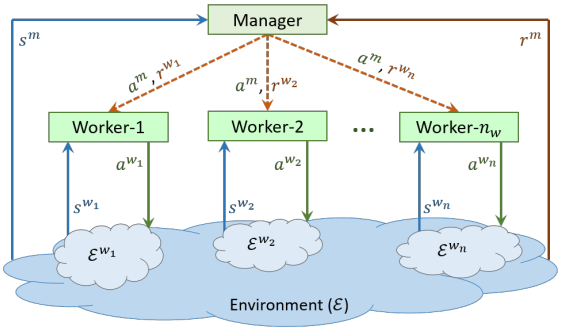

Fig. 2: Proposed multi-agent hierarchical reinforcement learning (MAHRL) model.

Manager. The state of manager involves the raw pixel values of the handwritten page image. The input image $\mathcal{I}$ is resized into size $n_z \times n_z$ by keeping the trace of the aspect ratio and produces the resized image $\mathcal{I}_z$. To maintain the aspect ratio, some columns or rows are filled with zeros. For our task, empirically, we choose $n_z = 1024$. The handwritten page images of our dataset are significantly large in size and are in grayscale. Such resizing assists in reducing the state space, but does not impede our objective. Moreover, the manager should have some perception about the text-line zones, since we aim to inspect the reading sequence at a text-line level. For text-line segmentation, we use an off-the-shelf semantic segmentation network [12], which is basically an encoder-decoder architecture, followed by a softmax layer to capture the pixel-level classification [13]. This network is pre-trained on small handwritten character graphemes to classify into ink-stroke region and background in order to fulfill the objective of text-line segmentation. Actually, the segmented mask ($\mathcal{I}_s$) is of size 1024×1024. At time step t, the manager also observes the workers' location coordinates on image $\mathcal{I}_z$. The workers' location matrix ($\mathcal{I}_l$) is also of size 1024×1024. Now, $\mathcal{I}_z$, $\mathcal{I}_s$, and $\mathcal{I}_l$ are composed as a single image ($\mathcal{I}_{zsl}$) comprising three channels, which is of size $1024 \times 1024@3$. This $\mathcal{I}_{zsl}$ is fed to a convolutional neural network (CNN) f_m to summarize the manager's state-space s^m. We obtain a feature vector v_m with dimension 256 from f_m. The f_m contains some sequentially added convolutional and pooling layers, as follows.

$\mathcal{I}_{zsl}\,(1024{\times}1024@3) \mapsto C_1\,(512{\times}512@16) \mapsto C_2\,(256{\times}256@32) \mapsto MP\,(128{\times}128@32) \mapsto C_3\,(64{\times}64@64) \mapsto C_4\,(32{\times}32@128) \mapsto MP\,(16{\times}16@128) \mapsto C_5\,(8 \times 8@256) \mapsto GAP\,(1 \times 1@256) \mapsto v_m$;

where, "C_i" $(\forall i = 1, 2, \ldots, 5)$ denotes the i^{th} convolutional layer, "MP" and "GAP" represent max-pooling and global average pooling layers, respectively [13]. The numeric values in the format of $(n_m \times n_m@n_c)$ symbolize the feature map size $n_m \times n_m$ and the number of channels n_c for a layer. For C_1 and C_2, we use 5×5 sized kernels, while for the rest of convolutions, 3×3 sized kernels are engaged. For all the convolutional layers, the stride size is 2. For max-pooling and global average pooling, the kernel sizes are 2×2 and 8×8, respectively. Here, each convolution is followed by a batch normalization [14] and a Mish [15]

activation function. The batch normalization is used to avoid overfitting. Mish has worked better than major state-of-the-art activation functions, e.g., ReLU, leaky ReLU, GELU, Swish [15], and it has also performed well for our task.

The manager decides which worker when to "*move*" or "*stop*", and commands the workers accordingly. Therefore, the action space (a^m) of the manager is discrete containing two actions (*move* and *stop*) per worker, i.e., a total $2n_w$ number of actions.

The manager focuses on the higher-level goal, and receives a reward $r^m = 1 - r_{LD}$, when a worker reaches at the end of a text-line; otherwise gets zero reward. Here, $r_{LD} = LD(\hat{l}_s, l_s)/max(|\hat{l}_s|, |l_s|)$ is a penalty term within the interval $[0, 1]$. $LD(.,.)$ is Levenshtein distance [16] that measures the distance between actual ($\hat{l}_s$) and predicted (l_s) word sequences.

At time step t, the manager observes the state s_t^m, and selects an action $a_t^m \in \mathscr{A}^m = \{1, 2, \ldots, |\mathscr{A}^m| = 2n_w\}$ to get a reward r_t^m. The fundamentals of reinforcement learning (RL) can be found in [6]. The manager learns a policy π_m to maximize the expected discounted return, which can be defined as the cumulative discounted reward, i.e., $\sum_{t>0} \gamma_m^t r_t^m$. Here, γ_m is a discount factor. To know how good the manager is learning over the policy π_m, values of the state (s^m) and the state-action pair (s^m, a^m) can be defined, which are called the *value* function (V^{π_m}) and the *Q-value* function (Q^{π_m}), respectively [6]. $V^{\pi_m}(s_t^m) = \mathbb{E}[\sum_{t>0} \gamma_m^t r_t^m | s_t^m, \pi_m]$; $Q^{\pi_m}(s_t^m, a_t^m) = \mathbb{E}[\sum_{t>0} \gamma_m^t r_t^m | s_t^m, a_t^m, \pi_m]$. The optimal Q-value function (Q^{*_m}) can be iteratively learned via deep Q-learning [5, 17], and the optimal action (a^{*_m}) is computed as follows. $a^{*_m} = \arg\max_{a\prime^m \in \mathscr{A}^m} Q^{*_m}(s^m, a\prime^m)$. Here, $Q^{*_m}(s^m, a^m) = \max_{\pi} Q^{\pi_m}(s^m, a^m)$. A deep Q-network $Q^m(s^m, a^m; \theta^m)$ with parameters θ^m can be employed to approximate the value functions [17]. In this paper, the manager adopts the concept of *dueling deep Q-network* due to its better performance than experience replay and prioritized replay-based architectures [18]. Here, a notion of *advantage function* (A^{π_m}) exists, which signifies how much an action is better than the expected. $A^{\pi_m}(s^m, a^m) = Q^{\pi_m}(s^m, a^m) - V^{\pi_m}(s^m)$.

The dueling deep Q-network architecture contains two parallel streams (*value*: f_{mv} and *advantage*: f_{ma}) of fully connected layers. The 256-dimensional feature vector v_m obtained from f_m is now fed to f_{mv} and f_{ma} streams in parallel to produce the separate estimates of the value (V^{π_m}) and advantage (A^{π_m}) functions. The f_{mv} comprises a fully connected layer with 128 nodes followed by ReLU activation, and a sequentially added fully connected single node to produce the output from the value stream. Similarly, f_{ma} contains a fully connected layer with 128 nodes trailed by ReLU activation, and another successive fully connected layer with $|\mathscr{A}^m|$ nodes to obtain the output from the advantage stream. Finally, two streams are combined to produce the Q-value function, i.e., $Q^{\pi_m}(s^m, a^m; \theta^m, \theta^A, \theta^V) = V^{\pi_m}(s^m; \theta^m, \theta^V) + A^{\pi_m}(s^m, a^m; \theta^m, \theta^A)$; where, θ^A, θ^V are the parameters of two sequences f_{mv} and f_{ma}, respectively. To tackle the identifiability issue [18] and to increase the optimization stability, this equation is modified as follows. $Q^{\pi_m}(s^m, a^m; \theta^m, \theta^A, \theta^V) = V^{\pi_m}(s^m; \theta^m, \theta^V) +$

$(A^{\pi_m}(s^m, a^m; \theta^m, \theta^A) - \frac{1}{|\mathscr{A}^m|} \sum_{a_\prime{}^m} A^{\pi_m}(s^m, a_\prime{}^m; \theta^m, \theta^A))$. The parameters $\{\theta^m, \theta^A, \theta^V\}$ are learned by standard policy-based RL strategy [19].

Worker. As mentioned before, while the manager focuses on the higher-level goal, the workers emphasize lower-level fine control to achieve sub-goals. A worker (worker-i) can observe the environment/handwritten page partially ($\mathcal{E}^{w_i}$), which formulates its state space (s^{w_i}). To express this partial observation, we perform a foveal transformation, as follows.

Foveal transformation (ϑ). A worker agent partially concentrates on $\mathcal{I}_z$ and extracts fragmental information around a location l. We here utilize the idea of foveated imaging [20,21], where the agent focuses at l with foveal vision, corresponding to the highest resolution; and with peripheral vision, as it gradually moves away from l by a lower resolution.

To encode the fragment region, we execute a foveal transformation $\vartheta(\mathcal{I}_z, l)$, which extracts the k (> 1) number of neighboring patches of different resolutions around location l. The foveal transformation is presented in Fig. 3(a). The 1^{st} patch is of size $w_p \times w_p$, 2^{nd} patch is of size $(w_p + d_w) \times (w_p + d_w)$, and so on, the k^{th} patch is of size $(w_p + (k-1).d_w) \times (w_p + (k-1).d_w)$, where d_w is the additional width of the successive patches. All the k patches are resized to $w_p \times w_p$, and thus produce k channels each having $w_p \times w_p$ sized patch-image. As a matter of fact, if l is near the boundary of $\mathcal{I}_z$, then peripheral patches are duly filled with zeros. Resizing a larger size patch to a smaller one lowers the resolution. In our task, empirically, we fix $k = 6$, $w_p = 64$, $d_w = 32$. Therefore, at time step t, by gazing at location l_{t-1} of image $\mathcal{I}_z$ and performing a foveal transformation $\vartheta(\mathcal{I}_z, l_{t-1})$, the agent produces a $64 \times 64 \times 6$ sized fragment p_t. The location l is encoded with a real-valued coordinate (x, y), where $0 \leq x, y \leq 1$. Here, the top-left and bottom-right coordinates of $\mathcal{I}_z$ are $(0, 0)$ and $(1, 1)$, respectively. In our task, the manager initializes the location l of a worker agent on the leftmost ink-stroke pixel of a text-line mask (with prior knowledge from $\mathcal{I}_s$) corresponding to the page image $\mathcal{I}_z$. A worker comprises fragment network (f_g), core network (f_h), and location network (f_l). The inside view of a worker is shown in Fig. 3(b), and its workflow is discussed as follows.

Fragment network (f_g). At time step t, a $64 \times 64 \times 6$ sized fragment p_t is inputted to a deep neural architecture f_g to extract features g_t. Our f_g architecture adopts ResNet-34 [22] with some minor amendments. The details of f_g are shown in Table 2, where the building residual blocks [22] are shown in brackets with the number of stacked blocks. For example, in *conv*1 (first convolutional layer), the input p_t of size $64 \times 64@6$ is convoluted with 32 filters of size $3 \times 3@6$ to produce a $64 \times 64@32$ sized feature map. Here, "$@n_c$" represents n_c number of channels. We use $stride = 1$ in *conv*1, whereas $stride = 2$ in *conv*2_x, *conv*3_x, *conv*4_x, *conv*5_x to perform down-sampling. Here also, each convolution is trailed by batch normalization [14] and a Mish [15] activation function. Intending to use f_g as a feature extractor, we discard the last fully connected layer of ResNet-34, which turns f_g into a 33-layered architecture. From the *avg_pool* (global average pooling) layer of f_g, we obtain a 512-dimensional feature vector g_t by flattening, at time step t.

Table 2: Architecture of f_g

Layer name	Output size	33-layer
input	64×64@6	
*conv*1	64×64@32	$3 \times 3, 32$
*conv*2_x	32×32@64	$\begin{bmatrix} 3 \times 3, 64 \\ 3 \times 3, 64 \end{bmatrix} \times 3$
*conv*3_x	16×16@128	$\begin{bmatrix} 3 \times 3, 128 \\ 3 \times 3, 128 \end{bmatrix} \times 4$
*conv*4_x	8×8@256	$\begin{bmatrix} 3 \times 3, 256 \\ 3 \times 3, 256 \end{bmatrix} \times 6$
*conv*5_x	4×4@512	$\begin{bmatrix} 3 \times 3, 512 \\ 3 \times 3, 512 \end{bmatrix} \times 3$
avg_pool	1×1@512	

Core network ***(f_h).*** At this point, g_t is fed to the core network f_h. The worker agent needs to memorize the past explored fragment information, therefore, a recurrent neural network (RNN) is employed as f_h. The GRU (Gated Recurrent Unit) [23] is used here as an RNN unit, due to its similar performance compared to LSTM (Long Short-Term Memory) for our task, while having fewer learning parameters [24]. The f_h contains 256 GRU units. Here, the agent maintains an internal state to encode information about *where* it gazed as well as *what* was observed. This information is crucial in deciding the action and finding the next location. The internal state h_t at time step t is updated over time by f_h. The present internal state h_t is a function of the previous state h_{t-1} and the external input g_t, which is formulated by GRU gates, as follows. $h_t = f_h(h_{t-1}, g_t) = \Gamma_u * \tilde{h}_t + (1 - \Gamma_u) * h_{t-1}$; where, $\Gamma_u = \sigma(\ell inear(h_{t-1}, g_t))$, $\tilde{h}_t =$

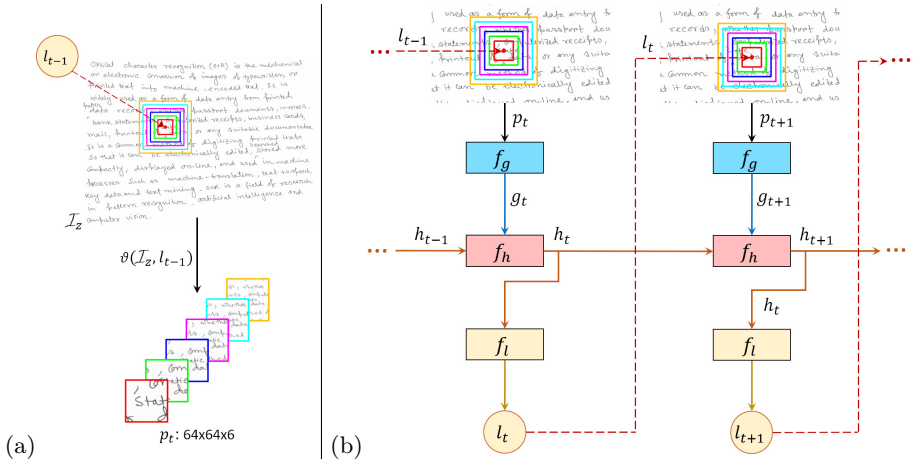

Fig. 3: (a) Foveal transformation $\vartheta(\mathcal{I}_z, l)$, (b) Internal workflow of a worker.

$tanh(\ell linear(\Gamma_r * h_{t-1}, g_t)), \Gamma_r = \sigma(\ell linear(h_{t-1}, g_t))$. Γ_u and Γ_r denote *update* and *relevant* gates of the GRU, respectively [23]. *Sigmoid* (σ) and *tanh* non-linear activation functions are used here [13]. $\ell inear(\bar{\mathrm{v}})$ represents the linear transformation of a vector $\bar{\mathrm{v}}$.

Location network (f_l). The h_t is embedded into f_l to find the next location l_t. The location policy is defined by a 2-component Gaussian with a fixed variance [25]. At time step t, we obtain the mean of the location policy from f_l, which is defined as $f_l(h_t) = \ell inear(h_t)$. Here, the fully connected layer is trailed by a sigmoid (σ) activation to clamp the location coordinates in $0 \leq x, y \leq 1$. The f_l is trained using RL [6] to find l_t for emphasizing to the next fragment p_{t+1}.

In RL, the worker agent interacts with the state s^w of the environment and takes action a^w to get the reward r^w. At time step t, the state s_t^w engages p_t around l_{t-1} and summarized into h_t. The action a_t^w at t is actually the location-action l_t chosen stochastically from a distribution θ_l-parameterized by $f_l(h_t)$. We shape the reward r_t^w at t internally, as follows.

$$r_t^w = \begin{cases} 1 - \alpha_d D^2(l_t, \hat{l}_t) - \alpha_y(y_t - y_{t-t_d}) & ; \text{if } t > t_d \\ 1 - \alpha_d D^2(l_t, \hat{l}_t) & ; \text{otherwise} \end{cases} \tag{1}$$

where, $D(.,.)$ is the Euclidean distance. The term $\alpha_d D^2(l_t, \hat{l}_t)$ signifies the loss due to the difference between actual ($\hat{l}_t := (\hat{x}_t, \hat{y}_t)$) and predicted ($l_t := (x_t, y_t)$) locations. The ground-truth was required here for the reward calculation. If the worker strays away from the designated horizontal text-line, then the manager penalizes the worker slightly with an amount of $\alpha_y(y_t - y_{t-t_d})$. Here, the hyper-parameters, i.e., $\alpha_d = 0.5$, $\alpha_y = 0.2$, and $t_d = 2$, are set empirically.

The RL-based agent learns a stochastic policy $\pi_\theta(l_t|s_{1:t}^w)$ at every t, which maps the past trajectory of the environmental interactions $s_{1:t}^w$ to the location-action distribution l_t. For our task, the policy π_θ is defined by early mentioned RNN, and s_t^w is summarized into h_t. The parameter $\theta = \{\theta_g, \theta_h\}$ is acquired from the parameters θ_g and θ_h of f_g and f_h, respectively. The agent learns θ to find an optimal policy π^* that maximizes the expected sum of discounted rewards. The cost function J_l is defined as follows. $J_l(\theta) = \mathbb{E}_{\rho(s_{1:T}^w;\theta)}[\sum_{t=1}^T \gamma^t r_t^w] = \mathbb{E}_{\rho(s_{1:T}^w;\theta)}[R]$; where, ρ is the transition probability from one state to another, which depends on π_θ [6]. T is the episodic time step and γ is a discount factor.

The optimal parameter is decided by $\theta^* = \arg\max_\theta J_l(\theta)$, where we employ gradient ascent using the tactics from RL literature [19], as follows. $\nabla_\theta J_l(\theta) = \sum_{t=1}^T \mathbb{E}_{\rho(s_{1:T}^w;\, \theta)}[R\ \nabla_\theta \log \pi_\theta(l_t|s_t^w)] \approx \frac{1}{N}\sum_{n=1}^N \sum_{t=1}^T R^{(n)} \nabla_\theta \log \pi_\theta(l_t^{(n)}|s_t^{w^{(n)}})$; where, trajectories $s^{w^{(n)}}$'s are generated by executing the agent on policy π_θ for $n = 1, 2, \ldots, N$ episodes. The $\nabla_\theta log \pi_\theta(l_t|s_t^w)$ portion is calculated from the gradient of RNN with standard backpropagation [11].

To avoid the high variance problem of the gradient estimator, variance reduction is performed here [26]. We employ variance reduction with the baseline ($\mathcal{B}$) that comprehends whether a reward is better than the expected one, as follows. $\nabla_\theta J_l(\theta) \approx \frac{1}{N}\sum_{n=1}^N \sum_{t=1}^T (R_t^{(n)} - \mathcal{B}_t) \nabla_\theta \log \pi_\theta(l_t^{(n)}|s_t^{w^{(n)}})$; where, $R_t =$

$Q^{\pi_\theta}(s_t^w, l_t) = \mathbb{E}[\sum_{t\geq 1} \gamma^t r_t^w | s_t^w, l_t, \pi_\theta]$ is *Q-value* function and $\mathcal{B}_t = V^{\pi_\theta}(s_t^w) = \mathbb{E}[\sum_{t\geq 1} \gamma^t r_t^w | s_t^w, \pi_\theta]$ is *value* function [6,26]. The learning of the baseline is performed by reducing the squared error between Q^{π_θ} and V^{π_θ}.

Finally, the detected series of the location (l_s) signifying the reading-sequence of words in a text-line, is reverted to the original input handwritten page image $\mathcal{I}$ by obtaining assistance from the previously-traced aspect ratio.

4 Experiments and Discussion

In this section, we present the dataset employed, followed by experimental results with discussions.

4.1 Database Employed

As we discussed earlier in Section 2, for the experimental analysis, we have procured 300 handwritten pages with the ground-truth information of the reading sequence of text-lines. Our database comprises 4 sets of data, i.e., DB_{IAM}, DB_{PBOK}, DB_{E}, DB_{B} containing 100 English, 100 Bengali, 50 English, and 50 Bengali pages, respectively.

Each dataset DB_{i} (for i $\in$ {IAM, PBOK, E, B}) is split into a training ($\text{DB}_{\text{i}}^{\text{tr}}$), validation ($\text{DB}_{\text{i}}^{\text{v}}$) and testing ($\text{DB}_{\text{i}}^{\text{t}}$) set with a ratio of 5:2:3. To reduce overfitting during training, we augment our training data $\text{DB}_{\text{i}}^{\text{tr}}$. For data augmentation, we randomly drop some word components from a handwritten page to dilute [27] the reading sequences. From a page, we generated 10 augmented pages.

4.2 Results and Evaluation

In this subsection, we present the experimental results to analyze our model performance. All results presented here were executed on the testing set $\text{DB}_{\text{i}}^{\text{t}}$ (for i $\in$ {IAM, PBOK, E, B}). The hyperparameters of our model were tuned and fixed during system training based on the validation/development set $\text{DB}_{\text{i}}^{\text{v}}$. Empirically, we set initial_learning_rate = 10^{-3}, discount_factor = 0.95, mini-batch_size = 32, episodic_time_step = 64, and episode = 512. We analyzed our model performance based on finding the location coordinates and subsequent detection of the reading sequence.

Location finding. The effectiveness of our model depends on finding the precise location to gaze at while reading. Here, we used the $RMSE$ (Root Mean Squared Error) [28] as a performance measure due to its efficacy in addressing the deviation of location-coordinate from the ground-truth (x_{GT}, y_{GT}). On a page, we computed the $RMSE$ over all the location coordinates. In Table 3, we present our model performance on location finding over $\text{DB}_{\text{i}}^{\text{t}}$ (for i $\in$ IAM, PBOK, E, B) in terms of $RMSE$ ($avg \pm sd$).

From Table 3, we can observe that the overall performance of location finding was the best for DB_{B} and the worst for DB_{IAM}. It is evident from this table that

Table 3: Performance on location finding

	$RMSE$ ($avg \pm sd$)			
Dataset	DB_{IAM}	DB_{PBOK}	DB_{E}	DB_{B}
OTL	0.02359±0.00347	0.01832±0.00127	0.01813±0.00753	0.01621±0.00283
non-OTL	0.01956±0.00147	0.01682±0.00358	0.01470±0.00098	0.01262±0.00489
Overall	0.02162±0.00433	0.01741±0.00445	0.01684±0.00564	0.01437±0.00389

Table 4: Performance on reading sequence detection

	MLD ($avg \pm sd$)			
Dataset	DB_{IAM}	DB_{PBOK}	DB_{E}	DB_{B}
OTL	1.4331±0.4745	1.1295±0.2387	1.1581±0.4462	0.8573±0.2740
non-OTL	1.0936±0.4033	0.9454±0.3631	0.8096±0.3400	0.7695±0. 3173
Overall	1.2673±0.8485	1.0330±0.6778	0.9870±0.8748	0.8156±0.5030

our system found the location better on unobstructed text-lines (non-OTL) than the obstructed text-lines (OTL).

Reading sequence detection. From the series of ground-truth coordinates (x_{GT}, y_{GT}), we can obtain the actual reading sequence $(\hat{l}_s)$ of a text-line. Similarly, from the predicted series of coordinates, we obtained the predicted reading sequence (l_s) of a text-line. Now, we measured the distance between the actual and predicted sequences with a small data-driven relaxation. Here, we used the Levenshtein distance (LD) [16] due to its efficiency in measuring the difference between the ground-truth and predicted sequences undertaking the pairwise sequence alignment. On a page, we computed the LDs for all the sequences over text-lines and took its page-level arithmetic mean, say, MLD. In Table 4, we present our model performance on reading sequence detection over DB_i^t (for i $\in$ IAM, PBOK, E, B) in terms of MLD ($avg \pm sd$).

From Table 4, we can see that our system performed better for unobstructed text-lines (non-OTL) than the obstructed ones (OTL) concerning the task of reading sequence detection. Here, the overall result was the poorest for DB_{IAM} as it contains the highest number of OTLs among the datasets employed.

In Fig. 4, we present some qualitative results of our system. The word insertions in OTLs of Fig. 4(b), (d) are done synthetically. Fig. 4(a) shows an example, where a single word is inserted by a caret after striking-out the mistaken word. In Fig. 4(c), two successive words are inserted by placing a single caret. Without any caret, a word is inserted in Fig. 4(b). In Fig. 4(d), two words are inserted separately by two respective carets, where our system fails to detect an insertion since the inserted word seems part of the prior text-line. Our system succeeds in Fig. 4(a), (b), (c).

4.3 Comparison

As we mentioned in Section 1, the approach undertaken in this paper is the earliest attempt of its kind, and we did not find any direct work for comparison

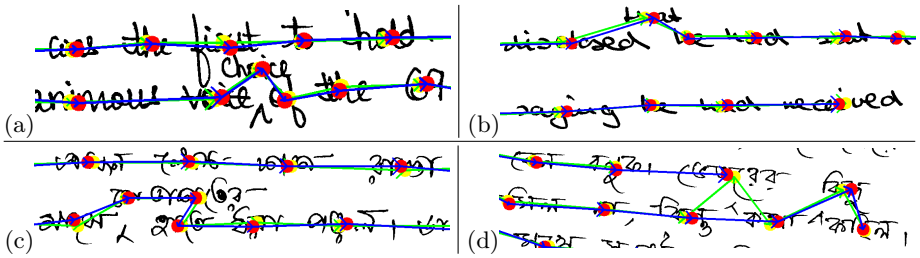

Fig. 4: Qualitative results of our system on samples of (a) DB_E, (b) DB_{IAM}, (c) DB_B, and (d) DB_{PBOK}. Detected locations and reading sequence paths are shown by red dots and blue arrowed-lines, respectively. Ground-truth locations and paths are also shown. (Softcopy exhibits better display.)

purposes. However, for comparison analysis, we adapted some indirect methods of the literature to fit into our problem, which is briefly discussed as follows.

Method-SN: We used SegNet [12] for semantic segmentation of both text-lines and words. Prior training was performed using small handwritten character graphemes. We obtained the CGs of the segmented words to treat as the location coordinates. The sequence of these locations was detected with a data-driven threshold.

Method-Ga: This method follows a similar approach to Method-SN in finding the location coordinates and subsequent sequence detection. The only difference is in the line and word segmentation module, for which we employed the 2D Gaussian filtering-based technique of GOLESTAN-a [29].

We also engaged some human knowledge to compare with our system.

Human-E: Here, 5 healthy persons without any known reading/writing disorders, were appointed to manually record the coordinates on the test data, similar to the ground-truth generation scheme (refer to Section 2). We took an average of location coordinates provided by 5 persons, which subsequently contributed to finding the reading sequence. All 5 persons appointed here had at least professional proficiency in the English language and they operated only on the English datasets, i.e., DB_{IAM}, DB_E. In this paper, we present the average result obtained from these 5 individuals.

Human-B: This setup is similar to the Human-E with the only difference is that all 5 persons appointed here had native proficiency in the Bengali language. The appointed persons here acted on Bengali datasets only, i.e., DB_{PBOK}, DB_B.

Human-nonB: This setup is similar to the Human-E and Human-B setups. The only difference is that all 5 adults appointed here had no proficiency in Bengali language. Still, they operated on Bengali datasets, i.e., DB_{PBOK}, DB_B. Here, the appointed individuals were provided some basic information of Bengali writing, such as Bengali is a left-to-right writing system similar to English, here too a word can be inserted with/without caret symbol, it maintains inter-text-line and inter-word gaps, etc.

Table 5: Comparative analysis

Method	Location finding				Reading sequence detection			
	$RMSE$ (avg)				MLD (avg)			
	DB_{IAM}	DB_{PBOK}	DB_{E}	DB_{B}	DB_{IAM}	DB_{PBOK}	DB_{E}	DB_{B}
Method-SN	0.04072	0.03908	0.03676	0.03530	2.3574	2.1206	1.8938	1.8370
Method-Ga	0.07427	0.07106	0.06861	0.06365	3.2776	3.0461	2.8664	2.6894
Human-E	0.00926	-	0.01038	-	0.0126	-	0.0065	-
Human-B	-	0.01161	-	0.01424	-	0.0232	-	0.0190
Human-nonB	-	0.01057	-	0.01591	-	0.8779	-	0.5603
Ours	0.02162	0.01741	0.01684	0.01437	1.2673	1.0330	0.9870	0.8156

As a matter of fact, the linguistic/readability knowledge of human-beings was used in the Human-E and Human-B setups, whereas such knowledge was missing in the Human-nonB setup. Owing to our limited opportunity, we were unable to make a rational choice of the experiment, i.e., Human-nonE, where the appointed individuals do not have any expertise in English/Latin script.

In Table 5, we compare our overall results with the performances of some baseline methods and humans, with respect to the location coordinate finding and reading sequence detection, while keeping the same experimental setup as before. For both cases, our system performed better than Method-SN and Method-Ga. From Table 5, it is interesting to observe that for a human (Human-E/Human-B) having linguistic knowledge, MLD $(avg) \approx 0$; but without linguistic knowledge, even human (Human-nonB) performance is not sufficiently reliable in finding the reading sequence. However, our system attempts to learn the reading sequence without any linguistic knowledge through the exploration-exploitation mechanism of RL.

4.4 Limitation

In this current research, we considered word insertion in principal text-lines [2], and did not tackle the (oriented) insertions in the margins. However, our work can be extended to address this issue. We here worked with obstructed text-lines at the word-level. Sometimes, in a word, character-level insertion/annotation can be noted, which we did not handle here.

We did not pay additional attention to the words with strike-through. However, a special module [30] for this can be added. Experiments were performed on handwriting samples written on white pages. However, if a sample is written on a rule-lined page, a preprocessing module [31] can be added.

We experimented on left-to-right writing systems (English and Bengali); however, our architecture can be extended to right-to-left writing systems [31] with some minor changes. Some unconventional annotations, due to writing idiosyncrasies, such as inserting a word below a certain text-line using a down-caret symbol ($_\vee$), are out of the scope of this paper.

5 Conclusion

In this paper, we studied the detection of the reading sequence of words in a handwritten text-line, including those that are obstructed due to inserting some words during amendments. We proposed a multi-agent hierarchical reinforcement learning-based architecture for our work. For experimental analysis, we took English and Bengali offline handwriting. We compared our system performance with some baseline approaches and obtained encouraging outcomes. However, still, there is a room for improving our system performance, which we will endeavor to address in the future.

Acknowledgment

All the people who contributed to generating the database are gratefully acknowledged. The authors also heartily thank all the consulted linguistic and handwriting experts.

References

1. T. Grüning et al., "A Two-Stage Method for Text Line Detection in Historical Documents", IJDAR, vol. 22, pp. 285-302, 2019.
2. L. L.-Sulem, A. Zahour, B. Taconet, "Text Line Segmentation of Historical Documents: A Survey", IJDAR, vol. 9, pp. 123–138, 2007.
3. O. Surinta et al., "A* Path Planning for Line Segmentation of Handwritten Documents", ICFHR, pp. 175-180, 2014.
4. X Y. Li et al., "Script-Independent Text Line Segmentation in Freestyle Handwritten Documents", IEEE TPAMI, vol. 30, no. 8, pp. 1313-1329, 2008.
5. K. Arulkumaran et al., "Deep Reinforcement Learning: A Brief Survey", IEEE Signal Processing Magazine, vol. 34, no. 6, pp. 26-38, 2017.
6. R. S. Sutton, A. G. Barto, "Reinforcement Learning: An Introduction", 2nd eds., MIT Press, ISBN: 9780262039246, 2018.
7. Wilber, "GIMP 2.10.22 Released", 2020. Online: gimp.org (retrieved: 3 May 2021)
8. U. Marti, H. Bunke, "The IAM-database: An English Sentence Database for Offline Handwriting Recognition", IJDAR, vol. 5, pp. 39-46, 2002.
9. A. Alaei, U. Pal, P. Nagabhushan, "Dataset and Ground Truth for Handwritten Text in Four Different Scripts", IJPRAI, vol. 26, no. 4, #1253001, 2012.
10. Y. F.-Berliac, "The Promise of Hierarchical Reinforcement Learning", The Gradient, 2019.
11. D. Wierstra, A. Foerster, J. Peters, J. Schmidhuber, "Solving Deep Memory POMDPs with Recurrent Policy Gradients", ICANN, pp 697-706, 2007.
12. V. Badrinarayanan et al., "SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation", IEEE TPAMI, vol. 39:12, pp. 2481-2495, 2017.
13. A. Zhang et al., "Dive into Deep Learning", 2020. Online: d2l.ai (retrieved: 3 May 2021)
14. S. Ioffe, C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", ICML, pp. 448-456, 2015.
15. D. Misra, "Mish: A Self Regularized Non-Monotonic Activation Function", Paper # 928, BMVC 2020.

16. V. I. Levenshtein, "Binary Codes Capable of Correcting Deletions, Insertions, and Reversals", Doklady Akademii Nauk SSSR, vol. 163, no. 4, pp. 845–848, 1965.
17. V. Mnih et al., "Human-level Control Through Deep Reinforcement Learning", Nature, vol. 518, 529-533, 2015.
18. Z. Wang et al., "Dueling Network Architectures for Deep Reinforcement Learning", ICML, vol. 48, pp. 1995–2003, 2016.
19. R. J. Williams, "Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning", Machine Learning, vol. 8, no. 3-4, pp. 229-256, 1992.
20. B. A. Wandell, "Foundations of Vision", ISBN: 9780878938537, Sinauer Asso. Inc., 1995.
21. H. Larochelle, G. E. Hinton, "Learning to Combine Foveal Glimpses with a Third-Order Boltzmann Machine", pp. 1243-1251, NIPS, 2010.
22. K. He et al., "Deep Residual Learning for Image Recognition", CVPR, pp. 770-778, 2016.
23. K. Cho et al., "Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation", EMNLP, pp. 1724-1734, 2014.
24. J. Chung et al., "Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling", NIPS Workshop on Deep Learning, 2014.
25. V. Mnih et al., "Recurrent Models of Visual Attention", NIPS, pp. 2204-2212, 2014.
26. R. S. Sutton et al., "Policy Gradient Methods for Reinforcement Learning with Function Approximation", NIPS, pp. 1057-1063, 1999.
27. J. Hertz, A. Krogh, R. G. Palmer, "Introduction to the Theory of Neural Computation", CRC Press, 1991. DOI: 10.1201/9780429499661.
28. A. Botchkarev, "Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology", arXiv:1809.03006, 2018.
29. N. Stamatopoulos et al., "ICDAR 2013 Handwriting Segmentation Contest", ICDAR, pp. 1402-1406, 2013.
30. B. B. Chaudhuri, C. Adak, "An Approach for Detecting and Cleaning of Struck-out Hand-written Text", Pattern Recognition, vol. 61, pp. 282-294, 2017.
31. W. A.-Almageed et al., "Page Rule-Line Removal Using Linear Subspaces in Monochromatic Handwritten Arabic Documents", ICDAR, pp. 768-772, 2009.