Automated Transliteration of Late Egyptian Using Neural Networks

Serge Rosmorduc

LINGUA AEGYPTIA

JOURNAL OF EGYPTIAN LANGUAGE STUDIES

28

2020

Widmaier Verlag · Hamburg 2020

LINGUA AEGYPTIA – Journal of Egyptian Language Studies (LingAeg)
founded by Friedrich Junge, Frank Kammerzell & Antonio Loprieno

LINGUA AEGYPTIA (recommended abbreviation: *LingAeg*) publishes articles and book reviews on all aspects of Egyptian and Coptic language and literature in the narrower sense:
(a) *grammar*, including graphemics, phonology, morphology, syntax, semantics, pragmatics, lexicography; (b) *Egyptian language history*, including norms, diachrony, dialectology, typology; (c) *comparative linguistics*, including Afroasiatic contacts, loanwords; (d) *theory and history of Egyptian literature and literary discourse*; (e) *history of Egyptological linguistics*. We also welcome contributions on other aspects of Egyptology and neighbouring disciplines, in so far as they relate to the journal's scope.
Short articles on grammar and lexicon will be published in the section "Miscellanies". Authors of articles or reviews will receive electronic off-prints. Periodically, we would also like to put the journal at the colleagues' disposal for a forum in which an important or neglected topic of Egyptian linguistics is treated at some length: in this case, a scholar who is active in this particular area will be invited to write a conceptual paper, and others will be asked to comment on it.
Authors should submit papers electronically to the managing editor (lingaeg@uni-goettingen.de). Please send contributions in both doc/docx and pdf format. Further information (incl. guidelines and a template) is available from www.widmaier-verlag.de. The decision whether to publish a manuscript is taken by the editors in agreement with the advisory board.

Addresses

Departement Altertumswissenschaften: Ägyptologie, Universität Basel
Petersgraben 51, 4051 Basel, Switzerland
Institut für Archäologie: LB Archäologie und Kulturgeschichte Nordostafrikas, Humboldt-Universität zu Berlin
Unter den Linden 6, 10099 Berlin, Germany
Seminar für Ägyptologie und Koptologie, Georg-August-Universität Göttingen
Kulturwissenschaftliches Zentrum, Heinrich-Düker-Weg 14, 37073 Göttingen, Germany
Institut für Ägyptologie, Universität Wien
Franz-Klein-Gasse 1, 1190 Wien, Austria

The annual subscription rates are 49 € for individual and 69 € for institutional subscribers while single issues are available for 99 € (incl. German VAT, excl. shipping). Orders should be sent to the publisher:
Widmaier Verlag, Kai Widmaier, Witthof 23F, 22305 Hamburg, Germany (orders@widmaier-verlag.de).

www.widmaier-verlag.de
ISSN 0942-5659 | ISBN 978-3-943955-68-2
DOI: 10.37011/lingaeg.28

CONTENTS

ARTICLES

REVIEW ARTICLE

REVIEWS

LingAeg 28 (2020), 233–257

Automated Transliteration of Late Egyptian Using Neural Networks

An Experiment in "Deep Learning"

Serge Rosmorduc[1]

Abstract

We apply Deep Learning techniques to the task of automated transliteration of Late Egyptian. After a brief presentation of the technology used, we examine the result to highlight the capabilities of the system, which is able to deal with a wide range of problems, including grammatical and phraseological ones. We then proceed to extract signs values from what the system has automatically learnt.

1 Introduction

This paper presents an automated transliteration system. As there are many transliteration styles in Egyptology, we will define what the system is expected to do. We aim to reproduce the kind of transliteration found in most philological works, and exemplified, for instance, in the grammar of François Neveu (Neveu 1996). Our system will not *directly* produce an analysis of the sign values. As we use Machine Learning techniques, the system will be dependent on its training corpus, extracted from the *Ramses Project*, and will reproduce the style of transliteration used there. In particular, the transliteration will be highly "normalising". It will outline the grammatical analysis of the sentence, and occasionally supply word endings or grammatical elements which have fallen from use by the end of the XXth dynasty, even when they are not written.

For quite some time now, we have been interested in automated machine transliteration of Egyptian; in 2005 we implemented a proof-of-concept system (Rosmorduc 2008), but

1 Laboratoire CEDRIC, Conservatoire National des Arts et Métiers, Paris (serge.rosmorduc[at] qenherkhopeshef.org).
This work would not have been possible without the Ramses corpus. I especially want to thank for their fine comments Jean Winand and Stéphane Polis, Mark-Jan Nederhof, along with the anonymous reviewers of this paper. The data used to produce this system, alongside a working trained python implementation of the current transliterator, is available at https://gitlab.cnam.fr/gitlab/rosmorse/ramses-trl.The data is expected to improve, both because of revisions of the Ramses corpus itself and of the code we use to extract the transliterated corpus. Those improvements will also be released as new versions of the corpus, with a different version number. A computer-science oriented article has been written and will eventually be available through the gitlab site.

DOI: https://doi.org/10.37011/lingaeg.28.07

it required a large number of expert-created rules to be efficient. Those rules were often conflicting: signs could have multiple values, multiple segmentations were possible… Solving those conflict on a large scale in a reliable way was extremely difficult.

Thanks to advances in machine learning, we present a new system which can handle Late Egyptian texts quite efficiently. One of its most interesting features is that it is able to somehow "explain" its behaviour, by using what is called an *attention* mechanism. This article will not only evaluate the system, but also highlight some of the most interesting linguistic features the machine has learnt.

The approach taken here is to consider *transliteration* as a kind of *rewriting* task. An *input text*, which is the *Manuel de Codage* encoding of the original Egyptian document, is rewritten as an *output text,* which is the transliteration. There are now many Machine-Learning tools and algorithms for rewriting tasks. Those tools have originally been developed for *automated translation*, a task far more complex than transliteration. Computer scientists have applied them to a wide variety of problems where the output is a text which is *somehow* a rewriting of the original, notably in summarisation tasks (Nallapati *et al.* 2016) and syntactic analysis (Vinyals *et al.* 2015).

Availability of large corpora is an adamant prerequisite for most modern machine learning algorithm. In this respect, the *Ramses corpus* (Winand *et al.* 2015, http://ramses.ulg.ac.be) is a solid ground on which to build such a system.

After a brief state of the art about automated transliteration, we give a short overview of neural networks, their use in Natural Language Processing, we present our corpus, and discuss our results.

2 Previous works

The earliest attempts at automated transliteration date back to the 1990's, with the thesis of Sophie Billet (1995; Billet *et al.* 1994). These early attempts, as well as those from the present author (Rosmorduc 2008; Barthélemy *et al.* 2011), were mostly based on hand-written rules, and tested on relatively small corpora.

Those handcrafted approach suffer from the difficulty to deal with conflicting rules in a reliable way, for instance to choose between various values for a given sign. Our own system relied on priorities given on rules and signs values. But the ultimate choice for a transliteration involves many different levels: sign values, possible sign combinations, the actual vocabulary, the syntax of the text, its semantics, and even phraseology. Balancing all of them by a trial-and-error approach can lead to good results on a limited corpus but is somehow doomed to fail for random texts.

It was thus reasonable to try to use machine learning techniques, which allow a system to compute its parameters from a corpus. Mark-Jan Nederhof and Fahrurrozi Rahman (2017) took the first steps in this direction by proposing a transliteration formalism based on statistical rules. However, his system required a specifically annotated corpus to train.

At that point, we were considering the use of Machine Translation techniques for transliteration. We had the *Ramses* corpus to work on, and soon, the publication of very efficient algorithms decided us to try this approach.

3 Corpus

The sophisticated algorithms of machine learning would be of little use without data to feed them. The availability of corpora is of prime importance. In our case, our participation in the Ramses Project (Winand *et al.* 2015, http://ramses.ulg.ac.be) of the University of Liège provided us with what is probably the largest collection of annotated *hieroglyphic* texts available on computer today.

The Ramses corpus is an extensive annotated corpus of Late Egyptian, which has been developed since 2006. It provides, for each text, the hieroglyphic transcription of its words, and their analysis as inflections of specific lemmas. Containing more than 500 000 words, the corpus is large enough for deep learning methods.

Obviously, as the texts are in Late Egyptian, a system trained on it will be biased toward this stage of the language and its graphic peculiarities. However, it also contains a number of monumental texts, including some from the eighteenth dynasty, which cover a bit of Middle Egyptian language and orthography.

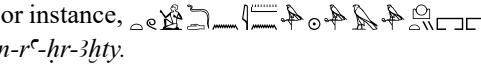
As machine learning needs a large volume of texts, we have chosen to use the whole Ramses corpus. It includes both the texts which have been made available on the Ramses Website, and which have been cross-checked and carefully proofread, and the texts which are yet to be validated. Most of those are of very high quality, but a few errors may occur here and there. It will lead to interesting cases later, where the automated transliteration system can actually improve some of the analysis.

The text transliteration has not been recorded in Ramses. Fortunately, each word is normally annotated with references to a spelling, a lemma, and usually to a specific inflexion of this lemma. In the database lexicon, "canonical" transliterations are provided for both spellings, lemmas and inflections. Hence, we can generate an *artificial* transliteration, which is the one we use later in the training process. For each word:

- if an inflection is specified, we use its transliteration;
- if a lemma is specified, but no inflection, the lemma's transliteration is used;
- if only a spelling is specified, the spelling's transliteration is used.

As a result, our transliteration is highly normalised. A word will be transliterated the way it "should" be written more than the way it is actually written. For instance, the infinitive of the verb *jrj* will be transliterated *jrj.t* even when it is simply written  .

This approach gives bad results in a few cases. For instance, as all occurrences of the preposition m are grouped together (same lemma, same inflection), spellings like  are transliterated as *m* and not *jm*.

The Ramses corpus also supplies prepositions when they are omitted in the first present, sequential, or third future. As we specify that those are "editor additions", they will not appear in the hieroglyphic transcription of the text, but they will be present in the transliteration, between ecdotic marks. For instance,  will be transliterated *twj* (*ḥr*) *ḏd n jmn-rˁ-ḥr-ȝḫty*.

As the system will be trained to produce transliterations by using the corpus as a sample, it will learn to supply missing prepositions, thus providing a rough morpho-syntactic analysis of the texts.

When we create the transliteration corpus, the words are kept in the same order as in the hieroglyphic text. It means that honorific transpositions are ignored, save in proper names and the like, where the transliteration is extracted as-is from the lexicon. This shortcoming is linked to the building of our particular corpus, and not to the machine learning process.

3.1 Corpus organisation

The first problem to solve was to choose *how* the corpus would be prepared for the machine learning task. We have decided to work on the sentence level, which is large enough to be useful, and small enough to be tractable on relatively small computers.

When working on Machine learning tasks, the standard approach is to cut the corpus into three sub-corpora:

- a *training* corpus on which the actual learning is done;
- a *validation* corpus which we explain below;
- a *test* corpus, which will be used to evaluate the results.

The system is repeatedly trained on sentences from the training corpus. The difference between the computed result and the expected output ends up as a numerical error. This error is used to gradually modify the system's parameters, improving again and again its results. However, with this criterium alone, a system which would learn to transliterate exactly its training corpus, no less, no more, would be considered perfect, while it is in fact useless. Thus, training algorithms try to avoid "rote learning".

One way to detect rote learning is to run the system on a different corpus, the *validation* corpus. This first evaluation shows how the system performs on texts outside its training corpus, measuring its capacity to *generalize* what it has learnt. To improve performances on the validation corpus, the computer scientist might change the learning algorithm. Its use to tweak the learning process, however, makes the validation corpus unsuitable to evaluate how the system would perform on random texts, and, in particular, unsuitable to compare the performances of two different systems.

The *test* corpus, in turn, solves this problem. While not used during the training process at all, the sole purpose of this corpus is to provide a final evaluation, in particular when comparing different machine learning approaches. It is technically sound to analyse the results obtained on the validation corpus to modify our training algorithms; but in theory the machine learning specialist should not even look at the test corpus content.

To build our training, validation, and test corpus, we had to decide how to dispatch our data. It seemed ill-advised to divide the sentences from a given text between the three corpora, as ideally, those corpora should be completely separated. The same word, occurring in two lines of the same text, is likely to have the same orthography and the same transliteration. This would likely result in over-optimistic evaluation of the learning quality.

Hence, we have assigned each text in the database to exactly one of the sub-corpora. We have randomly assigned 200 texts to each of the validation and test corpus, and 4403 texts

to training. The respective sizes have been chosen on relatively arbitrary ground. Besides, we have kept the same proportion of hieroglyphic texts and hieratic texts in each sub-corpus (7% of hieroglyphic texts and 93% of hieratic texts, corresponding to the current ratio in the Ramses corpus). The resulting training corpus is 1426499 sign long. From a statistical point of view, this approach is still somehow problematic, as it supposes that our texts are a representative random sample of the Late Egyptian language, which is not true: the ratio of Deir el Medina texts, for instance, is very high. For machine learning purpose, we do not have a good solution to this problem, as the original corpus is still relatively small, and that furthermore, whole areas of the country are completely absent from it. It would be, however, possible to evaluate our system on sub-corpora, both on geographic and chronological ground.

Each sub-corpus is made of two files: a source file, which contains the hieroglyphic texts, represented as lists of Gardiner codes (plus a few codes for lacunas), and a target file, containing the transliterations. The sentences are listed in a random order, one sentence per line, a line in the source file corresponding to a line in the target file.

For instance, these two lines[2] from the source test corpus:

N28 D36 D36 V31 S34 N35 Aa1 G24A Z2
M17 G17 G17 D36 D4 X1 Z7 V31A M17 M17 D21 T25 D58 Z7 Y1 I9

corresponds to the transliterations in the target corpus:

x a a _ = k _ a n x _ r x y . t _
i m y _ i r y . t w _ k y _ r _ D b A _ = f _

and to the hieroglyphs:

In the corpus, large lacunas for which no content is supplied are indicated with the code “LACUNA” instead of a Gardiner glyph code; words in lacuna which have been restored by the encoder are indicated with the code “MISSING”.

We have not kept the Manuel de Codage position codes, such as ‘*’, ‘:’ and ‘-’, in the source file. They can be useful, as word limits *tend* to occur at quadrant breaks. However, in the Ramses corpus, each word is encoded separately. As a result, the glyphs positions at the beginning and end of each word are unsure. Using them in training would lead the system to systematically consider that word limits are aligned with quadrant limits.

3.2 Evaluation Criteria

To assess the quality of the system, we need to check if transliterations produced by the system on the test corpus are correct. However, two transliterations of the same text are very unlikely to come out exactly identical, even when made by two expert human philologists.

2 The lines in the corpus are shuffled, so those two are taken from different texts (K*RI* 5, 15, 7 and K*RI* 1, 325, 4 respectively).

Machine Translation specialist evaluate their system by comparing each translation to multiple human-written "gold" translations. As the test corpus must be large enough to be reasonably significant, it requires a considerable amount of work.

As transliteration is somewhat simpler than translation, we have chosen to consider the transliteration in the test "target" file as the correct, "gold" transliteration. It is a gross approximation: some correct results will likely differ from the content of the "gold" corpus. However, the size of the data used in Natural Language Processing is so large that we have to make some compromises.

It would be overly pessimistic to assess the system quality by counting the number of identical lines in gold standard and the generated result. Instead, we resort to the system used in spell checkers: the *Levenshtein distance.* It's simply the number of characters which must be modified in the computed transliterations to obtain the gold standard. A perfect match will give a distance of 0; if the expected result was *sw ḥr stp*, and the computed result was *sww ḥr stp*, we would need to remove one of the "w" to get the correct result, which would give a distance of 1.

To get a coherent result on the corpus, given that the sentences have different lengths, we divide this distance by the length of the gold sentence, to get an "average number of edits by characters". For the previous example, we would then divide the distance by 9, the length of the expected result (including spaces) and get 1/9.

The variant of the distance we use is also somehow rough, as it considers only character insertion and removal, which means that a character replacement like *jst* vs. *jst* will end up as a distance of two edits.

The final result will be a mean evaluation over the whole corpus, which we hope to get well below 1.

4 Machine Learning and Neural Networks

Natural language processing, has come a long way since its beginning in the late 50's. In particular, the 1990's have seen a shift from hand-tailored formal system toward increasing use of data-intensive techniques, based on statistical models. The last ten years have seen a huge breakthrough with the use of Neural Networks, under the name of *Deep Learning*.

First used mainly for image processing, modern neural networks, thanks to increased computing power, much larger corpora, and theoretical improvements, have proved very efficient for many standard Natural Language Processing tasks, such as lemmatisation, named entities recognition, and machine translation.

4.1 A short overview of Neural Networks

Neural network are computer programs remotely inspired by the way actual neurons work. Figure 1, which shows a simple and classical system, will be used to explain their basic principles.

Let's suppose we want to recognise hieroglyphs. We have a picture of a sign, and we want to know to which Gardiner-code it corresponds.

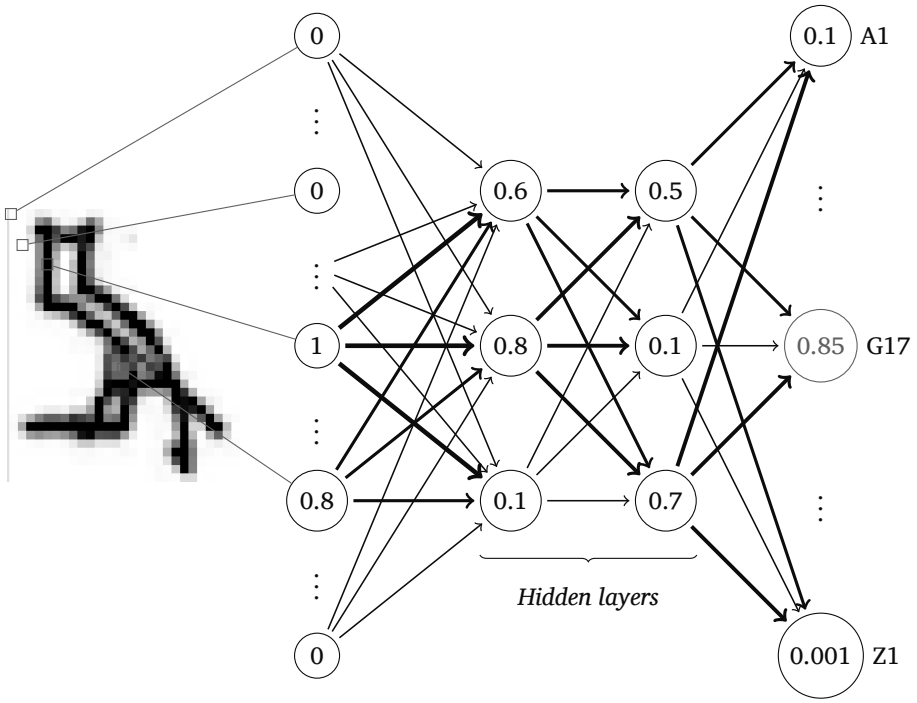


Figure 1 | A simple neural network

The neural network will *learn* how to do this from a large sample of already-labelled examples. The *input* is a numerical representation of the hieroglyph picture, and the *output* will represent the Gardiner code of the sign. The input would actually be a long list of numbers, which would correspond to the pixels in the sign's image. The output in this case would typically be another list of numbers as long as the number of Gardiner codes we would like to discriminate.

For each Gardiner code, the output layer provides a numerical value between 0 and 1, which indicates to what extent the original image depicts the sign associated with the Gardiner code. For instance, the value of 0.85 in Figure 1 indicates that G17 is a pretty good match for the picture, whereas the 0.1 value of code A1 tends to exclude 𓀀. In the final system, we might get intermediate result, which will indicate that the system hesitates between a number of similar signs.

Between the input and the output, we have the so-called "hidden layers". They simulate neural activation by assigning a numeric *weight* to each connection between two neurons, which indicate how strong this connection is and how much signal it conveys. The goal of the training process is actually to learn the best possible weights for the system. The computation of the *output* from the *input* obeys relatively simple mathematical laws. The system's complexity and ability to "learn" come largely from the sheer size of the network.

We would then *train* the network by presenting to it thousands of already identified examples. The weights are originally random values, so the first results are meaningless.

However, the system will adjust them to minimise the difference between the computed result and the expected one.

If both the training set and the network are large enough, training for a few hours or a few days will give a system which achieve good (but usually not perfect) performances. If properly trained, the system will be able to *generalise* from its input and will correctly classify data it has not already seen.

4.2 Encoder/Decoders and Attention

Text rewriting system, such as automated translators, or, in our case, automated transliterators, are a bit more complex than the system we have presented above. The main issue is that their input and their output have varying length.

4.2.1 Encoder and Decoder

The system we have used is called an Encoder/Decoder (Cho et al., 2014).

The *encoder* builds a numerical representation for each symbol in the input. Each Gardiner code in the input text will be represented as a list of numbers (usually a few hundreds). This representation is contextual: the list of numbers associated with a symbol will be influenced by the values associated with the neighbouring symbols (both before and after the sign).

The *decoder* generates the transliteration. We start with a special "begin of transliteration" character (let's say '#'), and then, one character at a time, knowing both the encoded hieroglyphic input and the already-generated transliteration, it will compute a probability for each existing transliteration character; we will usually choose the one with the higher computed probability.

For instance, if the input is  , and we have already generated the text "*#sw*", the "space" character, *i.e.* a separation between words, is most likely to occur next.

Once we have computed the next character, we feed it in turn to the decoder, and thus, character by character, we compute the "most likely" transliteration[3]. We stop when we predict a special character we have chosen as "end of sentence".

The whole process of generating the result one character at a time might seem very local. Yet, the system we built handles problems which can require to use information from the whole sentence to be solved. A number of architectural features in the network allows this. As the encoder is "bidirectional", i.e. the representation built at a given sign position depends on both the previous and the following signs, each encoder output depends on the whole input sentence. This capability is further improved by the attention mechanism we are about to discuss.

4.2.2 Attention

When the encoder/decoder computes the best value for the next sign, it needs to use a simple representation for the whole *input*. The original version of the encoder/decoder architecture uses the last encoder value. But it does not perform well on long sentences.

3 It's not mathematically true but will do in this presentation.

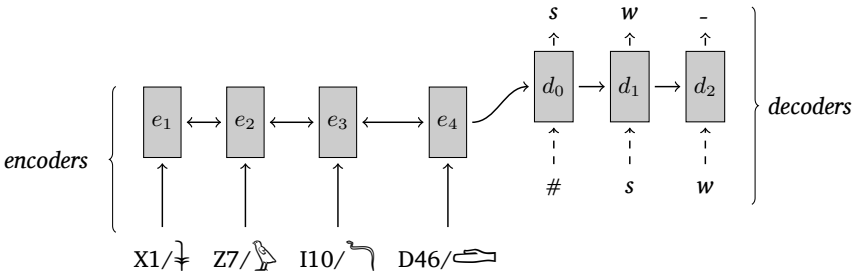


Figure 2 | Encoder/Decoder

As a result, a new mechanism, called *attention,* has been introduced (Bahdanau *et al.* 2015). When predicting the next sign in the output, it gives a weight, from 0 to 1, to each sign in the input. This weight captures the relevance of the hieroglyphic sign regarding the transliteration letter about to be generated. This attention is also learnt; it's not something defined beforehand.

In Figure 2, the source text is . If we have already generated the beginning of the output, "s,w", and try to generate the next character, our current system gives an attention of 0.85 on , of 0.09 on ; and 0.06 on the "sentence begin" code ('#' in the figure). The rest of the signs, (and), have negligible attention values. In a way, the system, which "knows" it has already generated *sw*, concentrates its attention on the glyphs at the beginning of the sentence (including the "#" which precisely indicates their start-of-sentence position). It then decides that *sw* is probably a word by itself and proposes a space as the next character.

If we go one set further, the system gives a very heavy weight (0.991) to , and actually propose a "(" as the next most likely output character following the space. In the end, we correctly generate the transliteration "*sw* (*ḥr*) *ḏd*".

Attention mechanisms have two interesting features: they improve the accuracy of the transliteration system, and they provide an insight into the inner working of the system, mitigating the "black box" aspect of neural networks in general.

5 Results

As we have explained above, the corpus is divided into three parts:

- a training corpus, on which the actual automated learning takes place;
- a validation corpus, which is used to evaluate if the learning has really improved, or if the system is doing rote-learning on the training corpus;
- a test corpus which is used to compare the respective performances of different learning systems.

The quantitative results below have been computed on the test corpus, and we will discuss qualitative results from the validation corpus. We made a number of experiments, using various systems and architectures, including OpenNMT and Tensor2Tensor. The best result corresponded with our own implementation of encoder/decoder with general global

attention, following Luong *et al.* (2015). We used Python 3 and the Keras framework. The hidden layers are 500 neurons wide; the network has 8,693,256 weights and uses 101 megabytes of memory. A detailed description of our system will be given in a technical article.

5.1 Quantitative Results

The test corpus is 2728 sentences long. The results for our best output on the test corpus give us an average edit distance of 0.094. It means that if we make an automated transliteration of a Late Egyptian text, we should expect to edit approximatively one letter out of ten.

The system has found the gold transliteration (0 error) for 1246 sentences – a little less than half of all sentences. A more in-depth study of the error distribution shows that the system performs better on the average for sentences whose transliteration is between 25 and 60 character long and degrades slightly with shorter or longer sentences.

If we take into account the various writing systems, the average Levenshtein distance is 0.092 on hieratic texts and 0.111 on hieroglyphic texts. This was expected, as hieratic has more redundancies the system can use, and the current corpus is richer in terms of hieratic texts.

5.2 Qualitative Results

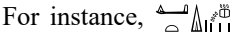
To stay in line with our principles, the qualitative results are extracted from the *validation* corpus.

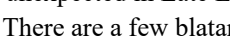
The longest sentence which was transliterated with 100% accuracy (that is, in complete agreement with the transliteration built from the corpus analysis) is the following (P. BM 10685, v° 2,1):

With the transliteration:

ḥry sꜣw.w sš.w jmn-ḥtp n šnw.wt pr-ꜥꜣ ꜥ.w.s n sš pꜣ-n-tꜣ-wr.t n tꜣ ḥw.t nswt bjty wsr-mꜣꜥ.t-rꜥ-stp.n-rꜥ ꜥ.w.s m pr jmn m ꜥ.w.s m ḥs.t jmn-rꜥ nswt nṯr.w.

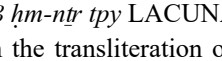
Many small differences between the generated text and the "gold" corpus are linked with different conventions in word endings.

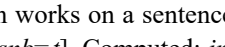
For instance,  (O. DeM. 852, ro, 1,1) has the "gold" transliteration *rdj.t dj.w n ꜣbd 3 šmw rdy.t bnr.w*, while the system produces: *rdj.t djw n ꜣbd 3 šmw rdy.t bnr*. A look at the lexicon for the Ramses database shows that both singular and plural spellings for *djw* and *bnr.w* have been recorded with a Z2 𓏺𓏺𓏺 ending - which is not unexpected in Late Egyptian.

There are a few blatant errors, as:  (O. Turin N 57001, r° 2), Gold: *nꜣ nty jb tḥ*; Computed: *nꜣ nty jw.t* (sic).

We have explained above that a large part of the corpus used for this experiment used texts from the Ramses database which have not been fully proofread yet. In quite a few cases, differences between the "Gold" and the "Computed" texts points toward an error in the initial analysis, or to a problem in the text itself.

In the case of  , (O. DeM 571, r° 5) "Gold" is *jw=w ḥr ꜥš pꜣ ḥm-nṯr tpy LACUNA*, whereas the computed version restores a missing *n*: *jw=w ḥr ꜥš* ***(n)*** *pꜣ ḥm-nṯr tpy* LACUNA.

In the transliteration of P. BM 10335, the system corrects the current analysis, and renders  as *twtw jw r=j mtr.t* instead of the erroneous "gold" *twtw* (*ḥr*) *jy.t r=j mtr.t*[4].

In most case, the sentences for which the computed sentences differ a lot from the original one contain a lot of lacunas. Their content is often restored from the surrounding texts in the gold transliteration, but this context is not available for the current system, which works on a sentence basis. A simple example is  (P. BM. 10190, 4), Gold: *jmy* [*snb=ṯ*], Computed: *jmy* [*ꜥnḫ=k*]. Here the process is simple: the sequence of wishes usual in a Late Ramesside Letters entails that the lacuna must be filled with *snb=ṯ,* whereas the system, which does not know the context at all, proposes a sequel for *jmy*, which accidentally almost hits the nail.

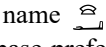
To see what kind of errors the system can make, and what it does well, we can have a look at one of the "worst" lines in *absolute* number of edits (K*RI* 4, 434,3):

The "gold transliteration" is here

sꜣ-wꜣḏyt nḫt-sw ḥy-nfr jmn-m-jn.t ***bw-ḳn.tw.f wꜣw-rn-f*** *ḫꜥ-****m-dwꜣw*** *nb-nfr ꜣny ḳḥ ḫꜥ-m-nwn jmn-m-ḥb* ***jpy*** *ḫꜣm smn-tꜣ.wy* ***gs wr*** *60* LACUNA

whereas the computed result is:

sꜣ-wꜣḏyt nḫt-sw ḥy-nfr jmn-m-jn.t ***ḳn.tw.f wꜣ rn=f*** *ḫꜥ-****sbꜣ-****nb-nfr ꜣny ḳḥ ḫꜥ-m-nwn jmn-m-ḥb* ***jpwy*** *ḫꜣm smn-tꜣ.wy-****m-wꜣst*** *60* LACUNA

This case is complex, because the document is a list, written with very abbreviated forms. In some cases, our rendering is better than the "Gold" transliteration, for instance for *jpwy.* In the case of the name *ḳn.tw.f* vs. *bw-ḳn.tw.f,* the system stays closer to the actual spelling. The name  is also probably more *ḫꜥ-*(*m*)*-sbꜣ* than *ḫꜥ-m-dwꜣ:* the Deir-el-Medine database prefers the former, and a quick search in the whole Ramses corpus (in which both transliterations have been used) reveals that the name never takes a determinative like ⊙, which would be expected for the word *dwꜣw*, *morning.* The system is obviously wrong in other cases: it links *ḫꜥ-sbꜣ* to *nb-nfr* as a single name, and makes two words of *wꜣ rn=f.*

4 Of course, both versions, including currently the "gold" one, lack an "m" in front of *mtr.t*.

The signs  at the end of the text, hint to a number of problems in the current data. First, those signs are actually left unanalysed in the Ramses corpus. No lemma is attached to them: the transliteration *gs wr* is in fact an artefact of the system we used to produce the corpus for this study, and not the responsibility of the encoder (K*RITA* suggests *m jꜣw* here).

For this text, the way we have prepared the corpus discards some interesting information, in particular about text layout and line endings on the manuscript. In the particular case of lists, which are often tabulated, this information is quite relevant. The line break before entails that those signs do not belong to the proper name  in the previous line.

One can also remark that the system can deal with the so-called "group-writings" reasonably well (see the spelling of the proper name *ḫḥ,* for instance). Each group is assigned a consonantal value, and, in the cases where determinatives occur at the end of the group they are not confused with word-endings. As the computed results imitates its training corpus, it does not attempt to restore any vowel.

5.3 Subject-Verb Agreement

The system "learns" at a global level. It takes a sentence in hieroglyphs and produces a transliteration. In this process, information is spread over the numerical values computed by the network. It encompasses both sign-level information and more global sentence-level knowledge.

The tools used by linguists, as paradigmatic replacement, allow to explore what the system knows – and what it has been able to generalise.

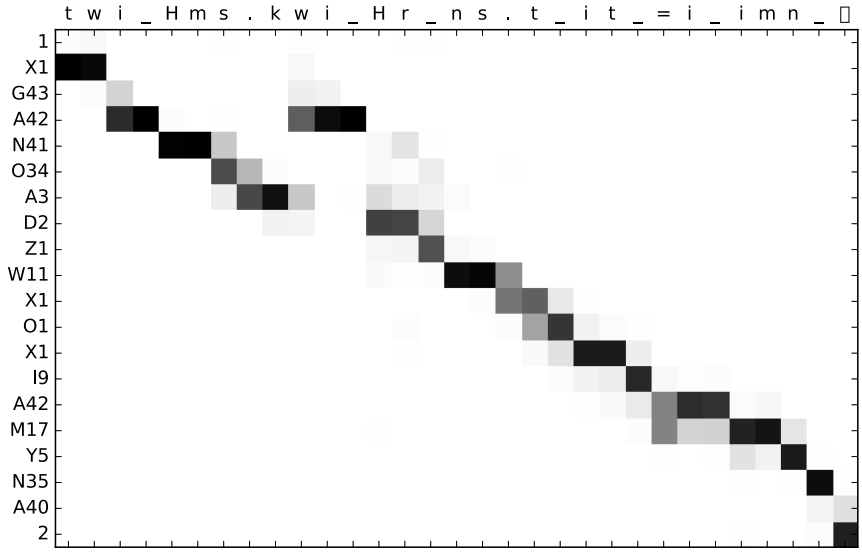
One very interesting point is that the network is able to "learn" some grammatical features, or, to be more precise, that somehow its internal representation captures a number of features we interpret as grammatical. To demonstrate this, we have tested the system with sentences where parts of the grammatical morphemes are missing from the hieroglyphic input.

We get the following results:

 *sw ḥms ḥr ns.t jt<=f> jmn*
 *twj ḥms.**kwj** ḥr ns.t jt=j jmn*
 *twk ḥms.**tw** ḥr ns.t jt=k jmn*

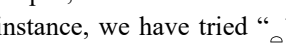
The system has correctly supplied old perfective inflection for the verb *ḥmsj,* which shows that it has some representation of expected verb forms (a first present would also have been possible), and of subject-verb agreement. The missing inflections are not written between parentheses as we might expect, as "*twj ḥms*(*.kwj*)", for instance, because in the current training corpus parentheses are only used to bracket whole words. Missing parts of word are silently restored.

Attention values for the first-person sentence (*twj ḥms.kwj*) are displayed in Figure 3. The interesting point there is that, when emitting the old perfective inflection "*.kwj*," the system attention is focused on the group X1-G43-A42, that is . The focus is even particularly strong on the sign A42.



(darker means stronger attention)

Figure 3 | Attention for omitted inflections

Now, all the old perfective forms above are documented in the corpus for the verb *ḥmsj*. But it we wanted to know if it worked for some forms unknown in the training corpus, which would mean that the system is able of some kind of generalisation. For instance, we have tried "  ". The resulting transliteration is "*twj nfr.kwj m pꜣ j.jr n=j nb*", and the attention structure, likewise, focuses on the first present preformant.

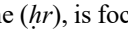
One caveat, however. Neural networks are not learning "rules". If one considers the analogy between them and biological cortex, the closest analogy might be the first layers of neurons dedicated to vision. In reality, they simply compute a bunch of numbers, and optimise their weight to minimize a computed loss, but we might metaphorically think of them as modelling a kind of "instinctive" language processing. No formal reasoning is used to choose between the various possibilities.

It means that sometimes, very close examples might give very different results. For instance, the first version of the network we had built for this article processed correctly examples built on the verb *ḫꜥj,* "to appear". With our latest network, whose results are overall better, the old perfective inflection was not restored for the *ḫꜥj* in the second person, and, for the first person, it was only restored when the first present preformant *twj* was spelt as  and , but not .

5.4 Deciding between old perfective and infinitive

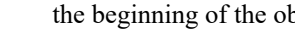
The system also captures information about specific verbs, and their possible environments. For instance,  will give *twk* (*ḥr*) *wnm db.w,* supplying the preposition

ḥr. This time, the system, instead of choosing an old perfective as in the previous examples, correctly supplies an expletive (*ḥr*). between parenthesis.

In this case, the attention information is also interesting. The attention, when producing the (*ḥr*), is focussed on the whole segment  , and in particular on the verb *wnm* itself. It *might* suggest that the system is taking into account both the verb, its subject, and the beginning of the object.

5.5 Text completion and phraseology

As, in the *input* corpus, some of the text are damaged, the system has some limited facilities for completing missing words or signs.

One of the most interesting examples we have found in the corpus is the following: 

(K*RI* IV, 266, 5). The "MISSING" codes are used when a word in the corpus is not associated with a spelling[5]. The generated transliteration is "*jw=tn ḥr ḏd n pr ꜥ3 ꜥ.w.s. pȝy=tn šrj nfr m mn.t*". The result is not correct, but is relatively reasonable, linguistically and philologically speaking. Our system has supplied a word for each MISSING element and provided a coherent grammatical sentence. We can guess that it has extrapolated *ḥr ḏd n* because of the epistolary formula found, for instance in O. DeM 327, 3: *r-ntt twj ḥr ḏd n jmn m-mn.t "I say to Amun daily."*

5.6 Proper name

In the corpus, long proper names are hyphenated. It allows the reader to distinguish them from normal sentences. The system learns to spot phrases which "look like" a proper name, and usually deals with them correctly. We wondered whether the names were somehow remembered, or if the system had developed a representation which would allow it to deal with new names.

To test this, we have tried it with an imaginary, but plausible name, which we inserted in a sentence: *  .

It is "correctly" transliterated as *ḏd=j n sš ḳdw.t* ***ḥr-m-pr-ȝs.t***. The proper name is completely made up, so it could not be learnt through the corpus.

The attention value when producing the first letter of the proper name is focussed both on the G5 Horus sign, and on the final determinative of the preceding title, *sš ḳdw.t.* This prompted us to check what would happen when removing the title. In this case, the system does not analyse *ḥm-m-pr-ȝs.t* as a proper name, and opts for independent words:

*  *ḏd=j n ḥr m pr ȝs.t.*

Thus, the choice to render a sequence of signs as a proper name is triggered both by its internal organisation, but also by the co(n)text.

One of the various networks we built had a very interesting bias (but we might be over-interpreting it): when faced with an unknown – but obviously – royal name, it would generate "*wsr-mȝꜥ.t-rꜥ*". This shows that the system weights both cotextual information,

5 This decree from Sethy II is not really damaged there; the missing spelling is an encoding error. The correct transliteration is *jw=tn* (*r*) *ḏd n=w ssnb pr-ꜥ3 ꜥ.w.s pȝy=tn šrj nfr*.

including what it has already generated, and local information. In this case, if the string of signs did not make a sense out of it, it would take the information from what was mostly expected after a *nsw.t-bjty* group.

5.7 Grammar, phraseology, context, and neural networks

As a partial conclusion, our system capabilities sometime suggest it has learnt something about text grammar. This leads to a number of questions. The first one is to decide what is the limit between grammar and phraseology. A much more thorough analysis would be needed, for instance, to determine if our system chooses to supply a "*ḥr*" before a verb Y because the context indicates that Y must be an active form (for instance, because it is followed by a direct object), or simply because the corpus contains more occurrences of "(*ḥr*) Y» than of old perfective Y.

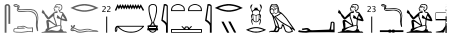
This being said, studies on the capabilities of neural network indicate that they can at least approximate some grammatical features. In particular Linzen (2016) has shown that a network could be taught to compute subject-object agreement in English, even when various substantives occurred between the subject and the object, which means the system was not simply taking the noun nearest to the verb as the subject.

6 Uses on Middle Egyptian

We have tested our system on the text of the *Shipwrecked Sailor,* with a mean error of 0.175. It is way larger than the one we got for Late Egyptian, but still reasonable. As It is possible to re-train an already trained network on different data, using a technique called *transfer learning,* using our current system as a starting point for training on Middle Egyptian corpora might be a valid approach.

If we dig deeper into our results, the bad score is caused by a few sentences with a very high error rate, whereas lots of other sentences are very well rendered.

For instance, l. 21–24 has 16 errors for an "gold transliteration" length of 62 (that is, 0.25 mean error):



"gold" transliteration: *sḏd=j rf n=k mjtt jry ḫpr.w m-ᶜ=j ḏs=j šm.kwj r bjȝ n jty*
computed transliteration: *sḏd=j r=f n=k mjt.t jry ḫpr* ***m-dj=j*** (***ḏs=j*** is missing) *šm.kwj r* ***bḥm*** *n* ***sbk***

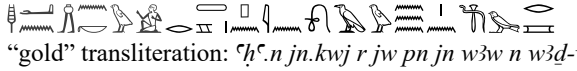
Among those differences, *m-dj* instead of *m-ᶜ* is expected, as both the Late Egyptian *m-dj* and the Middle Egyptian *m-ᶜ* have the same hieroglyphic spelling. Other differences are caused by encoding habits (*mjtt* vs. *mjt.t*), by gaps in the training corpus (*sbk* vs. *jty*)or are plain failure (the missing rendering of *ḏs=j*).

The system performs much better on the next sentence of the text, which is almost as long:


“gold” transliteration: *hꜣ.kwj r wꜣḏ-wr m dp.t n.t mḥ 120 m ꜣw=s mḥ 40 m wsḫ=s*
computed transliteration: *hꜣ**b**.kwj r wꜣḏ-wr m dp.t n.t mḥ **100 20** m ꜣw=s mḥ 40 m sḫ.**wt**=s*

Here, the main problem is the spurious “b” in the initial verb, which is not due to a lack of similar examples in the corpus, as the love song from O. DeM 1266 and O CG 25218, l 12 contains “*hꜣ.kwj r mw*” as a protasis.

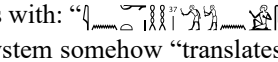
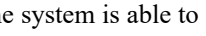
Systematic errors, based on the grammatical differences between Middle and Late Egyptian, occur as expected: the system often understands *sḏm n=f* instead of *sḏm.n=f,* for instance. It has also difficulties with old perfective after *ꜥḥꜥ.n* as in l. 109. It tends to interpret them as perfective *sḏm=f.*


“gold” transliteration: *ꜥḥꜥ.n jn.kwj r jw pn jn wꜣw n wꜣḏ-wr*
Computed transliteration: *ꜥḥꜥ.n jnj=**k wj** r jw pn jn wꜣw n wꜣḏ-wr*

Or l. 129:


“gold” transliteration: *ꜥḥꜥ.n sbꜣ hꜣ.**w***
computed transliteration: *ꜥḥꜥ.n sbꜣ **(ḥr) hꜣj.t***

In this last case, the system has favoured the Late Egyptian narrative pattern over the Middle Egyptian one.

Finally, l. 36, contains one of the most tantalising examples the system has provided us with: “  ” is rendered *m ḫt ḥw.t n=j sy*. It gives the impression that the system somehow “translates” Middle Egyptian into Late Egyptian. However, it is but an artefact of the way our corpus is built. The cleft-sentence markers *jn* and *m* are recorded under the same lemma in *Ramses*. As a consequence, when we have built the training corpus, all cleft-sentence initial *jn* were transliterated as *m.* It remains that the system has correctly identified the initial *jn* as a cleft sentence marker. To check this, we have tried to see if it would differentiate this *jn* from the homonymic question marker by testing the sentence “*  ”, which is correctly rendered as *jn jnk sš.* It demonstrates that the system is able to differentiate two kinds of sentence-initial , depending on the following context.

7 Learning Sign Values?

The attention values open an interesting window on the inner parts of the system. It was tempting to try to use them systematically. Thus, we decided to try to extract some kind of “sign values” from the attention system. We ran the system on the *training* corpus (because of its sheer size) and analysed each attention matrix. For each sentence, we extracted the following pieces of information:

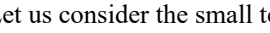
A. for each transliteration letter, which single hieroglyph reccind the most attention?
B. for each transliteration letter, which hieroglyphs, taken together, explain up to 99 % of the attention ?

To be more precise, for each transliteration letter, each hieroglyph in the input has an attention value, and the sum of the attentions is one. We pick the hieroglyphic sign with the largest attention value. This will be used in column A. Then, we pick the second largest attention value, and we continue until we have reached a total attention of 0.99. This is the basis of column B.

Note that in case A, we will ascribe each transliteration letter to a *single* hieroglyph. Also note that a hieroglyph may not be listed at all, if no attention is focussed on it.

The final result goes from hieroglyphs to transliteration. For each hieroglyphic sign in the input, we collect all transliteration letters for which this sign is in group A, and each transliteration letter for which the sign is in group B.

What we really compute is not strictly the sign value as a human reader learns it. A hieroglyphic sign will be associated with a transliteration letter when it is instrumental in the decision of choosing this specific transliteration letter. To be clearer, in some cases, the mere appearance of a  in a hieroglyphic text suffices to decide that we need to transliterate *ḏd*, without even looking at the following *d*; in this case, the whole attention for both letters of *ḏd* might be focussed on the single glyph . Hence, the notion of "value" used here is a bit far-fetched and different from the intuitive idea.

Let us consider the small text fragment:  "*t3 bw.t n p3 nṯr*". Table 1 describes the values we have extracted for each sign occurrence.

Sign												
A	*t3*	SEP	*bw*	.	*t*		SEP	*n* SEP	*p3*	SEP	*nṯr*	SEP
B				*w*	.; SEP	*t* SEP	*t*; *n*	SEP				

SEP *indicates word-breaks*

Table 1 | Signs values from attention

In the case of ideograms, this approach often gives their actual phonetic value. For instance, is correctly identified as *nṯr*: In general, this algorithm tends to segment the text in small groups whose signs are interpreted together, and to attach a transliteration to the first hieroglyph of the group. It works well for and *p3,* but tends to fail for uniliteral signs. Thus, *bw* in *bw.t* is mainly linked to , the *w* sign having only a secondary importance in the process.

Determinative and a number of uniliteral signs are also correctly identified as word endings (, and).

Appendix A lists the more common values for a number of signs. For a large number of signs, they can occur at the beginning or at the end of words, hence the large number of occurrences of SEP in the sign values.

For uniliteral signs, which often occur as "phonetic complements", their "real" value is generally found in list B. This list is built on the basis of the signs which are needed to explain up to 99% of attention, and not on the signs which receive the maximum of attention. As such, it's most appropriate for phonetic complements. For signs like  (M17), as the attention seems to concentrate on group beginnings, it captures in list **A** a number of groups which start with "*j*": *jw, jm.* The same is true for  or .

For a few signs, what we think of as their "standard" value is simply not listed in our top five values. For instance,  is correctly listed as "=*f*" when it occurs as a pronoun, but not as "*f*" when it occurs in the middle of words.

The system understanding of biliteral signs is more reliable. For most of them, their usual value occurs early in list A: for instance, the first values for  and  are respectively *wꜣ* and *šw*. For a sign like  , the first values are *pꜣ* and *wsf.*

Other kinds of signs are usually relatively well recognised.  and similar signs are understood either as first-person suffix, or as word limits (SEP), which is the expected value for determinatives. Likewise,  gains most of its attention at SEP word endings.

The sign  is understood as a word ending SEP, or as a writing of consonants *w, t* or *s.*

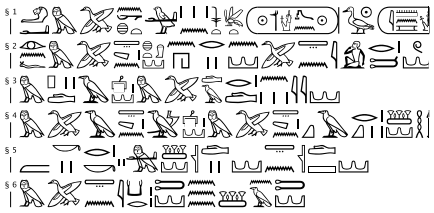
The dichotomy between  and  is quite interesting. In the database, it seems that the first is mainly a word ending (hence a determinative), and that the second, which is far less frequently encoded, mostly encodes either *jw* or *nm.t*.

8 Limits of the system

The system has a number of shortcomings. Some come from the corpus organisation, and others from the deep learning system.

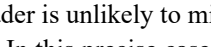
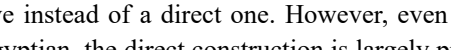
The first limitation is the sentence-based organisation of the corpus. Trained on whole sentences, the system will try to make a sentence with any text. For instance,   will be transliterated as *j.ꜣb,* considering the one-word sentence as an imperative. Now, this precise choice is quite reasonable, but will complicate some possible usages, like dictionary searches. Above all, the system will have difficulties when the text has not been segmented in sentences. Fortunately for us, human encoders are not always coherent in segmenting their texts, which means that the corpus contains sentences which are actually sequences of relatively independent propositions. The problem is also slightly mitigated by the presence in the corpus of damaged texts and of various kinds of lists.

More fundamentally, the encoder/decoder technology still builds a kind of fixed length representation of its whole input text: the data that the encoder sends in the decoder. As a result, the system memory can become relatively deficient on long sentences, especially if the said sentence has a relatively uniform structure. In the following extract from the poem of Qadesh, the decoder finds itself unable to keep track of the exact position in the input text and skips a whole passage.



The resulting transliteration is: (§ 1)

ḥ3t-ʿ m p3 nḫtw n nswt bjty wsr-m3ʿ.t-rʿ stp.n-rʿ s3 rʿ rʿ-ms-sw-mry-jmn dd ʿnḫ ḏ.t (§ 2) *jr.n=f m p3 t3 n ḫt3 nhrn p3 t3 n jrṯ* (§3) *m pds* (the system skips ***m p3 drdny (T 4) m p3 t3 n ms***) *m p3 t3 n ḳrḳš ḥnʿ rk* (§ 5) *m krkmš ḳdy t3 n ḳdš* (§ 6) *m p3 t3 n jkrt mšnt*

In other cases, some rather simple segmentation can be missed (K*RI* IV, 14,3–4)  is transliterated *j.dj ḥry.t n nḥḥ m* ***jbwš***, whereas the explicit addition of a genitive "n" enforces a better result:  *j.dj ḥry.t n nḥḥ m* ***jb n mšwš***. On the original text, not only did the system come up with a wrong segmentation, but it also completely neglected the groups , which a rule-based system or a human reader is unlikely to miss.

In this precise case, the system performs better if we provide it with an indirect genitive instead of a direct one. However, even if indirect genitives are more usual in Late Egyptian, the direct construction is largely predominant in the case of *jb.* The error made by the system cannot be blamed on the corpus content.

The most glaring problem is that the system can fail where a simple rule-based software would perform well. Its training enables it to use high level information, like morphology, syntax, and somehow phraseology, to transliterate a sentence. But on the other side, its understanding of the value of individual sign is limited. This behaviour is somehow useful for Late Egyptian, as it allows the system to ignore extraneous aleph, weak consonants, or space fillers. On the other side, it has a bothering habit of neglecting or misinterpreting some uniliteral signs. For instance, if the text to analyse is the single group  , it transliterates it as "LACUNA" (sic). On the other hand, is correctly rendered as *j.gr*: by interpreting the word as an imperative, the system makes a full sentence out of it and gives a reasonable transliteration.

Another type of errors a rule-based system would not make is the generation of apparently out-of-place parts of sentences. Transliterating the name , the system produces ***mry-rʿ*** *ḫʿ-ḫpr-rʿ-snb*, the initial ⊙being rendered twice.

9 Corpus size sensitivity

Most of the time spent in digital humanities, especially when applying machine learning algorithms, is devoted to the corpus preparation. Finding its minimal size is thus a reasonable concern.

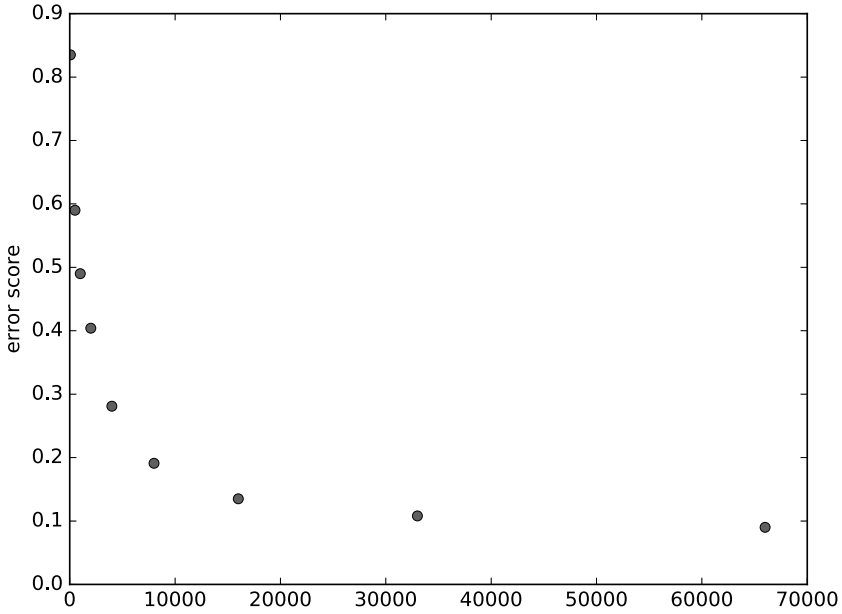


Figure 4 | Error score versus corpus size

We have trained the system on reduced corpora to evaluate the impact of corpus size, from 62 sentences (approximatively 1,300 signs) to 66000 (approximatively 1,400,000 signs). Figure 4 shows the corresponding *average levenshtein distance* scores. It seems that reasonably good results can be expected for a corpus of 16000 to 20000 sentences, or 340,000 to 400,000 signs, preferably from various types of texts. To give an idea of the minimal corpus size, it's 25 times as large as *Horus and Seth,* our longest literary text, and 17 times as large as P. BM. 10052, the largest text in Ramses.

10 Possible Extensions

10.1 Improvements of the current system

Extensions of and improvements to this system will come from two sources. First, from better algorithms: each year since 2014 has seen a small revolution in deep learning. Our current system is based on Luong (2015), which has since been outperformed by the transformer (Vaswani et al. 2017) and BERT (Devlin et al. 2019).

The other improvement will come from the data itself. The original corpus has lots of information we have not used. First, to overcome the "sentence boundaries" problem, we intend to produce a new corpus which will be made from arbitrary samples of texts, some spanning more than one sentences, some starting in the middle of a sentence, etc. The idea would be to enable the system to start at any arbitrary point in a text, and even segment the texts into sentences if needed.

A second improvement would be to intentionally introduce lacunas in the texts. This is a well-known technique in machine learning to improve the generalisation capabilities

of a system and to prepare it for "noisy" input. If we feed the system with a few more sentences with lacunary input, but full transliteration, we might expect it to improve its capacities at restoring part of a missing text. The exact amount of lacunary input which will give reasonable results is an open question.

Finally, it would be interesting to try to mix the current, black box, approach, with a rule-based system; in *Machine learning*, combining different systems, which make different errors, is a rather usual approach; however, in the present case (and in Natural Language Processing in general), there is no obvious way to do so.

10.2 Extension to other corpora

The present system is also a proof-of-concept. It proves that automated transliteration is possible and gives an estimation of the size of the corpus one needs to get reasonable results. The limitations linked with the use of the Ramses corpus regarding word boundaries or honorific transpositions could be lifted with a plain transliterated corpus.

For extending the present work to Middle Egyptian, a technique known as *transfer learning* (Goodfellow *et al* 2016) could possibly be used to limit the size of the necessary corpus, by reusing part of the current network. For Ptolemaic texts, more work would be needed, but the general principles used here would apply. We would be happy to work with colleagues who have those corpora in electronic form.

10.3 Extension to other tasks

The current system is very much a black box, even if the attention system allows one to get a partial understanding of how the final transliteration is produced. A number of early readers of this article have expressed their wish to get better information on each glyph value. Given a large enough annotated corpus, this would be a relatively easy task – annotating symbols in a sequence is a well-known application of neural networks. Learning it from our current corpus is much more complex and left for further explorations.

As the current system "learns" some grammar to an extent, it is also tempting to try to train it to tag individual words with their part of speech and inflections. Part of speech tagging is a classical application of Natural Language Processing, but it is usually performed with whole words as entries. Here, however, the entry would be a list of signs. Such systems automatically deal with orthographical variation, as they don't rely on a lexicon. They are not that widely used on modern languages, which have a relatively rigid orthography, because their performances tend to be lower when dealing with normalised text.

11 Conclusion

We have provided here a first usable corpus for automated transliteration. The results are very tantalising. Transliteration as such might not be the main issue for scholars, but it's a proof of concept for other applications: lexical studies, teaching aids, error detection in the corpus, to name a few. The main drawback of the current method is the huge amount of data needed. Further improvements might reduce this requirement.

Bibliography

Bahdanau, Dzmitry, Kyunghyun Cho & Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate, in: *3rd International Conference on Learning Representations, ICLR. arXiv preprint arXiv:1409.0473v7.*

Barthélemy, François & Serge Rosmorduc. 2011. Intersection of Multitape Transducers vs. Cascade of Binary Transducers: The Example of Egyptian Hieroglyphs Transliteration, in: *Proceedings of the 9th International Workshop on Finite State Methods and Natural Language*, Blois, France, 74–82.

Billet, Sophie 1995. *Apports à l'acquisition interactive de connaissances contextuelles*. Thèse de doctorat, Université Montpellier II.

Billet-Coat, Sophie & Danièle Hérin-Aime. 1994. A multi-agent architecture for an evolving expert system module, in: Dimitris Karagiannis (ed.), *Database and Expert Systems Applications*, Lecture Notes in Computer Science, Berlin & Heidelberg, 581–590.

Cho, Kyunghyun, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk &Yoshua Bengio. 2014. Learning Phrase Representations Using RNN Encoder–Decoder for Statistical Machine Translation, in: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Doha, 1724–1734.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee & Kristina Toutanova. 2019. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding, *ArXiv:1810.04805 [Cs].*

Goodfellow, Ian, Yoshua Bengio & Aaron Courville. 2016. *Deep Learning*, Cambridge, Massachusetts.

Linzen, Tal, Emmanuel Dupoux & Yoav Goldberg. 2016. Assessing the Ability of LSTMs to Learn Syntax-Sensitive Dependencies, in: *Transactions of the Association for Computational Linguistics* 4, 521–535.

Luong, Thang, Hieu Pham & Christopher D. Manning. 2015. Effective Approaches to Attention-Based Neural Machine Translation, in: *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, Lisbon, 1412–1421.

Nallapati, Ramesh, Bowen Zhou, Cicero dos Santos, Caglar Gulcehre & Bing Xiang. 2016. Abstractive Text Summarization Using Sequence-to-Sequence RNNs and Beyond, in: *Proceedings of the 20th SIGNLL Conference on Computational Natural Language Learning*, Berlin, 280–290.

Nederhof, Mark-Jan & Fahrurrozi Rahman. 2017. A Probabilistic Model of Ancient Egyptian Writing, in: *Journal of Language Modelling* 5 (1), 131–163.

Neveu, François. 1996. *La Langue Des Ramsès – Grammaire Du Néo-Égyptien*, Paris.

Rosmorduc, Serge. 2008. Automated Transliteration of Egyptian Hieroglyphs, in: Nigel Strudwick (ed.), *Information Technology and Egyptology in 2008: Proceedings of the Meeting of the Computer Working Group of the International Association of Egyptologists (Informatique et Egyptologie), Vienna, 8–11 July 2008*, Piscataway, 167–83.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser & Illia Polosukhin. 2017. Attention Is All You Need, in: *Advances in Neural Information Processing Systems* 30, 5998–6008.

Vinyals, Oriol, Łukasz Kaiser, Terry Koo, Slav Petrov, Ilya Sutskever & Geoffrey Hinton. 2015. Grammar as a Foreign Language, in: *Advances in Neural Information Processing Systems* 28, 2773–2781.

Winand, Jean, Stéphane Polis & Serge Rosmorduc (2015). Ramses. An annotated corpus of late Egyptian, in: P. Kousoulis (ed.), *Proceedings of the 10th International Congress of Egyptologists, University of the aegean, Rhodes 22–29 May 2008*, Orientalia Lovaniensia Analecta, Leuven, 1513–1521.

Appendix A. Selected Computed "Sign Values"

The following section contains a number of values for selected signs, using the A and B lists of values (section 7) The number after each value is the number of occurrences of this value found when transliterating the training corpus. For each sign, we have listed in A and B only the five most frequent values.

To avoid wasting printed space with an overlong list, we have picked for each type of signs a few representative cases. The full list is however available in the git archive of the project (https://gitlab.cnam.fr/gitlab/rosmorse/ramses-trl).

Uniliteral

 (G1) **A.** SEP (12737), *y* (5451), - (2650), *w* (1196), . (790)
B. *ꜣ* (5216), **SEP** (3483), . (2599), *t* (533), *n* (481)

 (M17) **A.** *iw* (6599), . (6038), **SEP** (5149), *im* (4510), *i* (3840)
B. **SEP** (10399), . (4717), *n* (3757), *w* (2800), *t* (1913)

 (D36) **A.** **SEP** (4219), . (1536), *ꜥ* **SEP** (989), - (599), *w* (478)
B. **SEP** (2702), *ꜥ* (1546), . (651), *t* (493), *n* (351)

 (W11) **A.** *gr* (371), *ns.* (216), *gꜣ* (159), *ꜣ* (110), **SEP** (109)
B. **SEP** (409), . (127), *g* (86), *w* (85), *t* (70)

 (D58) **A.** **SEP** (1513), *bn* (995), *bw* (963), . (631), *bi* (425)
B. *b* (1439), **SEP** (1115), . (303), *w* (255), *t* (164)

 (Q3) **A.** **SEP** (1959), *pt* (1066), . (965), *pn* (743), *pꜣ* (622)
B. **SEP** (1384), *p* (1314), - (596), *t* (483), *y* (277)

 (I9) **A.** *=f* **SEP** (12364), *f* **SEP** (2451), **SEP** (1296), *r* (713), **SEP** *=f* **SEP** (215)
B. **SEP** (3274), . (1304), = (1013), *r* (511), *r* **SEP** (471)

 (G17) **A.** *m* **SEP** (9227), *m-* (4618), **SEP** (2764), *y* (1061), *mt* (1009)
B. **SEP** (3684), *m* (2030), . (693), *t* (578),) (494)

 (F32) **A.** *ẖ.* (196), *ẖr* (101), *ẖd* (95), *ẖs* (82), *ẖꜥ* (22)
B. **SEP** (57), *t* (53), *ẖ* (27), . (13), *w* (13)

 (N37) **A.** *šm* (870), **SEP** (837), *šr* (335), *šꜥ* (278), *šf* (130)
B. *w* (628), **SEP** (422), *š* (415), . (116), *t* (104)

 (X1) **A.** *t* (8740), *t* **SEP** (7090), *.t* (6741), **SEP** (5208), *tꜣ* (3935)
B. **SEP** (22346), . (8120), *t* (6441), *t* **SEP** (3777), *w* (1733)

 (D46) **A.** **SEP** (5029), *ḏr* (1489), . (1160), *dm* (313), ((206)
B. . (1692), **SEP** (1480), *d* (762), *t* (443), *ḏ* (301)

 (I10) **A.** *ḏd* (3215), *ḏ.* (568), *d* (356), **SEP** (236), (*ḥr*) (206)
B. **SEP** (705), *ḏ* (260),) (131), *t* (81), - (78)

Biliteral

 (O29) **A.** ꜥ3 (1148), ꜥ3 **SEP** (1076), *pr* (395), ꜥ3. (250), *pr-* (114)
B. **SEP** (1255), - (716), . (328), ꜥ (153), *t* **SEP** (86)

 (O29v) **A.** *3-* (2), ꜥ3 **SEP** (2), **SEP** ꜥ3. (1), ꜥ3- (1), ꜥ3*y* (1)
B. - (61), *pr-* (32), *b-* (23), *b* (21), *n-* (13)

 (V4) **A.** *w3* (164), *3* (137), *š* (56), **SEP** (29), . (24)
B. *w* (142), **SEP** (120), . (51), *r* (20), *t* (20)

 (G29) **A.** *b3* (1068), *b3k* (88), *bḏ* (41), (*ḥr*) (37), *bn* (29)
B. *k* (891), **SEP** (149), *b* (37), - (29), *r* (25)

 (G40) **A.** *p3* **SEP** (1177), *p3y* (278), *p3* (243), *p3-* (55), **SEP** *p3* **SEP** (20)
B. **SEP** (305), *y* (85), - (37), *n* (29), . (27)

 (G41) **A.** *p3* (9588), *wsf* (444), <*n*> (115), *3* (104), **SEP** (64)
B. **SEP** (4193), *y* (1002), - (981), > (214), > **SEP** (185)

 (M16) **A.** *ḥ3* (520), **SEP** (65), *3* (58), *ḥ3* **SEP** (40), *ḥl* (27)
B. **SEP** (188), *w* (80), *y* (78), *ḳ* (65), *p* (43)

 (M12) **A.** *ḫ3* (647), *3* (356), *ḫb* (160), *ḫr* (123), *1000* **SEP** (122)
B. **SEP** (597), *0* (219), *00* **SEP** (195), *0* **SEP** (175), *000* **SEP** (77)

 (U30) **A.** **SEP** (252), *š* (89), *3* (22), . (20), *y* (5)
B. *3* (236), *3* **SEP** (129), *3.* (53), **SEP** (34), *t3* (14)

 (H6) **A.** *m3ꜥ.t* (125), *šw* (119), *m3ꜥ* (76), *m3ꜥ.* (41), **SEP** (27)
B. - (173), **SEP** (153), . (69), *.t* **SEP** (67), *t* (46)

 (H6A) **A.** *šw* (101), *šwy* (42), *3ꜥ* (7), **SEP** (4), *m3ꜥ* (2)
B. **SEP** (113), ꜥ (51), - (45), *y* (36), *pḥ* (12)

 (U23) **A.** *mḥr* (241), *mḥr* **SEP** (165), *3b* (128), *mḥ* (82), *r* (54)
B. **SEP** (300), *r* (67), *mḥr* **SEP** (62), *mḥr* (48), *r* **SEP** (41)

 (K1) **A.** *n* (42), *rmw* (7), *in* (3), *w* (2), *wn* (2)
B. *n* (95), **SEP** (26), *w* (23), . (16), *n.* (14)

 (D4) **A.** *ir.* (1655), *iri* (1224), *ir* (938), *iry* (511), *ir* **SEP** (391)
B. **SEP** (1450), . (1053), *i* (281), *i.* (197), *w* (135)

Selected other signs

 (Ff1) **A. SEP** (3900), *w* (735), *t* (533), . (528), *s* (348)
B. SEP (7423), . (1253), *w* (1136), *t* (1040), *s* (717)

 (Z1) **A. SEP** (17074), *1* **SEP** (3271), - (1513), **SEP** *1* **SEP** (1040), *7* **SEP** (754)
B. SEP (13073), *t* (4431), *n* (2003), = (1750), *w* (1126)

 (Z2) **A. SEP** (4293), *w* **SEP** (1316), *5* **SEP** (1077), *3* **SEP** (955), **SEP** *3* **SEP** (794)
B. SEP (7963), *w* (1485), *n* (1436), *w* **SEP** (1251), . (1174)

 (V20) **A.** *10* **SEP** (2828), **SEP** (1853), *0* **SEP** (1376), *2* (1338), *20* (570)
B. SEP (11652), *0* **SEP** (4925), *0* (1947), *s* (317), **SEP** *60* (230)

 (A1) **A.** =*i* **SEP** (4037), **SEP** (2259), *i* **SEP** (1189), *w* **SEP** (443), . (239)
B. SEP (6178), . (771), *n* (667), *w* **SEP** (489), *w* (485)

 (A2) **A. SEP** (1418), . (565), *t* **SEP** (291), *n* (206), *w* **SEP** (120)
B. SEP (1400), *i* (1223), *r* (608), *t* (604), . (526)

 (A4) **A. SEP** (31), . (3)
B. SEP (9), *w* (3), *.w* **SEP** (3), *w* **SEP** (3), **SEP** = (2)

 (A5) **A. SEP** (1), *imn* (1), *imn* **SEP** (1)
B. SEP (4), *t* (2), = (1), - (1), *13* (1)

 (A24) **A. SEP** (721), *nḫt* (512), *t* **SEP** (172), *w* **SEP** (108), - (104)
B. SEP (2235), - (456), . (362), = (245), **SEP** *m* (220)

 (A42) **A.** =*i* **SEP** (554), **SEP** (269), *i* **SEP** (74), *t* **SEP** (23), **SEP** =*i* **SEP** (13)
B. = (284), **SEP** (169), **SEP** = (53), *t* **SEP** (39), *i* (37)

 (B1) **A. SEP** (724), =*t* **SEP** (285), *t* **SEP** (205), *w* (108), *t* **SEP** (95)
B. SEP (872), *t* (585), . (317), *t* **SEP** (180), *n* (152)

 (D6) **A. SEP** (398), *ptr* (59), *ptr* **SEP** (51), . (21), *t* **SEP** (21)
B. SEP (510), **SEP** = (94), = (86), *n* (53), . (45)

 (D40) **A. SEP** (1017), . (175), *.t* **SEP** (132), *nḫt* **SEP** (110), *t* **SEP** (110)
B. SEP (1342), *w* (295), = (275), . (261), **SEP** = (216)

 (D54) **A. SEP** (982), *t* **SEP** (744), *.t* **SEP** (531), *iw* **SEP** (320), . (168)
B. SEP (1891), . (866), = (491), *r* (484), **SEP** = (351)

 (D54A) **A.** *iw* **SEP** (25), *nmt* (13), **SEP** (2), *iw* (2), *iw.* (2)
B. SEP (11), *.t* (7), - (2), *10* (2), *.t* **SEP** (2)

 (P5) **A.** *ḫw* (191), *nfw* (70), *ḫ* (45), *ḫw* **SEP** (38), **SEP** (24)
B. SEP (94), *w* (51), - (28), *t* (24), *t* (15)

 (P1) **A. SEP** (181), *t* **SEP** (69), *.t* **SEP** (63), *wi3* **SEP** (37), *imw* (34)
B. SEP (248), *i* (72), . (70), *n* (35), *t* (35)