Project Report

Automated Essay Scoring

IASD Master

Pierre-François Massiani
Guillaume Le Moing

work completed during the
Natural Language Processing class

April 25th , 2020

Contents

1 Introduction

This is the final project for the « Natural Language Processing » class of Master IASD (Artificial Intelligence, Systems, Data), a joint PSL University Master program hosted by Paris-Dauphine, École normale supérieure, and MINES ParisTech.

In this project we propose and compare different neural network architectures as well as learning strategies for the automated essay scoring task. We test our algorithms using the data from the ASAP competition on Kaggle [1] sponsored by The Hewlett Foundation.

Our project was implemented in Python. The code is available on a Github repository at this address:

`https://github.com/16lemoing/automated-essay-scoring/`

Instructions are included in the repository to run the code on your machine.

1.1 The automated essay scoring task

The task we are trying to solve here is called *automated essay scoring* (AES). It consists in automatically assigning a grade to an essay written on a given topic. This is of particular interest in an educational setting, where incentives for developing unified and objective grading methods can be easily understood. Such a method could be used for instance to grade large scale exams where teachers need to evaluate hundreds of essays in a short amount of time, leading to exhaustion, unwanted bias introduced by how focused the teacher is when reading, and discrepancies in the grades caused by the teachers' own evaluation criteria. Using an automated grading system can then help to standardize this process. For example, writing tasks were suppressed from French PACES[1] competitive exams this year because teachers would not have enough time to grade them because of the coronavirus outbreak [2]. An automated grading system could help to solve such problems. Other factors also account for the growing interest in this task, such as cost reduction.

In spite of these incentives pushing for the development of automatic essay scoring algorithms, the attempts at using them received quite a backlash. The arguments against these algorithms were that they did not understand the meaning of the essays they were grading and were relying only on "surface features of responses". To prove their point, protestors created essays exploiting biases discovered in the algorithm to create nonsensical high-graded

[1] "Première année commune des études de santé", the first year of health studies in France.

essays. According to MIT Director of Writing Les Perelman, "the substance of an argument doesn't matter [for such algorithms], as long as it looks to the computer as if it's nicely argued" [3]. Indeed, high grades were given to essays containing assertions that were simply not true, such as stating that the War of 1812 started in 1945.

The Hewlett Foundation challenge In 2012, *The Hewlett Foundation*[2] sponsored a competition on Kaggle [1] intended at demonstrating how AES algorithms - and more specifically neural networks - could be as reliable as humans in rating thousands of essays. Although this competition was very successful [4–7], it is still controversial whether the initial claim is backed up by the competition's results. In this project, we propose an algorithm for the Hewlett Foundation's challenge.

1.2 Related work

The interest for the AES task sparks around 1970 with the works of Ellis Batten Page [8], but it quickly faces the limitations imposed by the computational power available at the time. This practical limitation is lifted in the 1990s, and 1999 sees the first commercial automatic essay grader [9]. After that, the field grows rapidly and different approaches are tested with the state-of-the-art statistical inference techniques known at the time [10].

Today, the AES task is typically tackled using modern and state-of-the-art natural language processing (NLP) tools. Indeed, AES is a subtask of the more general text classification problem, which is a vibrant field in the NLP community. As such, classical NLP techniques can be used to achieve great results such as dense neural networks [11], convolutional networks [12], or recurrent networks [13].

2 A Neural Network based approach

We choose to rely on neural networks for this task. In this section, we first describe the dataset and then the whole preprocessing and architecture choices. We wanted to be able to test many possible choices: for each part of the processing pipeline, there are different options between which we will have to choose.

[2]The Hewlett Foundation is a private foundation that grants awards in a variety of liberal and progressive causes such as education, climate, health, journalism...

Essay set	Type	Set size	Average length	Range 1	Range 2
1	Argumentative	1785	350	1-6	2-12
2	Argumentative	1800	350	1-6	1-4
3	Source based	1720	150	0-3	-
4	Source based	1772	150	0-3	-
5	Source based	1805	150	0-4	-
6	Source based	1800	150	0-4	-
7	Argumentative	1730	250	0-15	0-30
8	Argumentative	918	650	0-30	0-60

Table 1: Some statistics about the essay sets. A source essay is provided for all "Source-based" essay. The scoring methods are quite complex and are described more thoroughly in Section 3.1. The scoring ranges provided are used for detailed scores or global ones.

2.1 Dataset

The dataset we work on can be downloaded from the competition's website [1], along with explanations on the essays and the (human) grading methods.

Essays description There are eight essay sets, each generated from a single prompt: each set was created by collecting essays from students who worked on a particular topic. Hence, topics are *coherent* among one set, and so is grading. The instructions across sets vary a lot : some essays are argumentative and should defend a point about something (e.g., "the effects on computers on people"), and others consist in text commentary. More details are given on each set in Table 1. All essays were written by Grade 7 to Grade 10 students, and hand graded and double-scored. What's more, essays are graded on different criteria and, depending on the set, a detailed score is given as well as how to compute the global score from the detailed one.

Dataset The dataset comes with the following fields:

- `essay_id` An essay identifier, unique for each essay
- `essay_set` 1-8, an identifier for each set of essays
- `essay` The text of the essay, encoded in ASCII
- Fields describing the detailed and global scores:

- `rater1_domain1`, `rater2_domain1`, `rater3_domain1` Domain 1 score for each rater
- `domain1_score` Resolved Domain 1 score
- `rater1_domain2`, `rater2_domain2` Domain 2 score for each rater [3]
- `domain2_score` Resolved Domain 2[4]
- `rater1_trait1_score` - `rater3_trait6_score` Trait scores[5]

The scoring fields are missing in the validation and test sets. However, we absolutely need them in order to first train, and then evaluate our models. Hence, we only work on the competition's training set and we divide it into our own training, validation and test sets. What's more, we do not follow the formatting of the output as it is described on the competition's website, since it is not required for this project.

A note on anonymization The essays may mention names, dates, locations, organizations, etc. Such mentions were replaced before the publication of the dataset with easily identifiable keywords to anonymize the essay: we will keep that in mind when designing our algorithm.

Additional features In the raw dataset, the only features we can use for prediction are the set number and the raw ASCII text. As described in Section 2.6, we start by enriching the dataset with extra features we compute from the text itself. The enriched dataset is generally denoted with the suffix `_x` in our files, and is the one we use for prediction.

2.2 Words embedding

Essays have to be turned into vectors of numbers before being fed to a neural network. One popular strategy is to assign a vector to every single word in the text. The required dimension for the vectors depends on the richness of the vocabulary (both in quantity and lexical diversity).

2.2.1 Random

This is the most basic embedding. We assign to each word of the vocabulary a vector of random samples from a normal (Gaussian) distribution.

[3] Only for Set 2.
[4] Only for Set 2.
[5] Only for Sets 7 and 8.

2.2.2 Word2Vec

We propose to train a Word2Vec model [14] on the sentences extracted from the training essays. This model learns meaningful embeddings by trying to guess a word from its context (a few words before and after in the sentence). To do this each word of the context is turned into a vector from which the prediction is made. Those vectors are what we use as embeddings once the model is trained. Each set of essays deals with a different topic. We hope to capture topic-related knowledge from the corpora by learning the embedding directly from the set of training essays.

2.2.3 GloVe

We also propose a different strategy for getting word embeddings using GloVe [15]. The particularity of GloVe embedding is that there are linear substructures between words sharing semantic aspects. For this embedding, we decide not to train the GloVe model from scratch but use a pre-trained model on large-scale corpora such as Wikipedia instead.

2.3 Essay preprocessing

We present here a few preprocessing methods that can help the learning process.

2.3.1 Spelling errors correction

Reading a few essays, we realised that a significant number of words were misspelled. This can impair the prediction performance because we cannot provide a meaningful embedding to misspelled words and end up with an artificially much larger vocabulary. It would have come in handy to have a python wrapper for a correction tool such as LanguageTool which handles both syntaxic and grammatical errors. Instead we used pyspellchecker package which gives a list of the most plausible correction candidates for each of the misspelled words (but do not consider the sentence as a whole).

2.3.2 Stopwords removal

There are some very common words, called *stopwords*, that usually do not add much meaning to the sentence. A common preprocessing step in natural language processing consists in removing these words that can pollute the learning process. However, in our case it is unclear whether we should remove these words or not. For example we need to take them into account when

evaluating the gramatical correctness of a sentence. That being said, it would be surprising that our model learns what is a grammaticaly correct sentence from such a small corpus.

2.4 Essay processing

We describe here how the essays are transformed so that they can be understood by a neural network.

2.4.1 Word tokenization

The first step is to transform essays into tokens (isolated words). To do that we transform every special character into a space symbol and then split the essay at every occurence of the space symbol.

2.4.2 Essay padded encoding

Then we assign to each word its index in the vocabulary. This index will be used as a key when retrieving the word embedding. To enable batch-learning we pad every essay so that it matches the length of the longest essay.

2.5 Multitask model

The dataset contains multiple essay tasks which are scored on different scales. This makes it difficult to learn all tasks jointly. However this can be addressed by normalizing the scores.

2.5.1 Score normalization

We rescale all the scores so that they fall into $[0, 1]$, by applying this simple linear transformation:

$$s_{norm} = (s - s_{min})/(s_{max} - s_{min}) \tag{1}$$

2.5.2 Score recovering

For the evaluation we need to recover the score in its original scale. To do this we apply the inverse transformation and then round the obtained value to the nearest score value in the corresponding set.

2.6 Extra-features computation

When predicting a score for an essay it is possible to include higher-level features to give some insights about characteristics that are hard to grasp for the neural network. First, during the spelling correction, we can compute the number of misspelled words for each essay. Doing this, the model can be fed the corrected essays and have a better understanding of the meaning while still being able to judge on this aspect. We also extract part of speech indicators from the essays (number of nouns, adjectives, verbs...), usage of ponctuation (number of question marks, exclamation marks, commas...), semantic diversity (number of semantic roots that were used to build the words in the essay), quotations (counting quotation marks, references to organizations, locations, people...).

Whether we include these extra features or not is a choice of architecture for the model, and will be one of the hyperparameters we will adjust through the validation procedure.

2.7 Models architecture

We propose highly customizable neural networks to optimize the final architecture based on the validation results.

2.7.1 Dense

First, we propose a four-layer dense neural network. The first layer is the embedding layer. We then take the mean of all the embedding vectors and feed it to a series of dense layers. We use ReLu activation functions except for the output layer which has a Sigmoid activation when the scores are rescaled to $[0, 1]$. Dropout is added between layers to prevent overfitting. The dense model can be fed either encoded essays alone, both encoded essays and extra-features, or extra-features alone. This way we can evaluate the predictive power of each individual part.

2.7.2 LSTM

The dense neural network merges all word embeddings into a single vector. It does not take into account the order in which the words appear in the essay. We introduce an LSTM model so that we can work directly with sequenced data. Our LSTM model is made of a custom number of layers and can include dropout. We also add a few fully connected layers so that we can make use of the extra-features if they are provided.

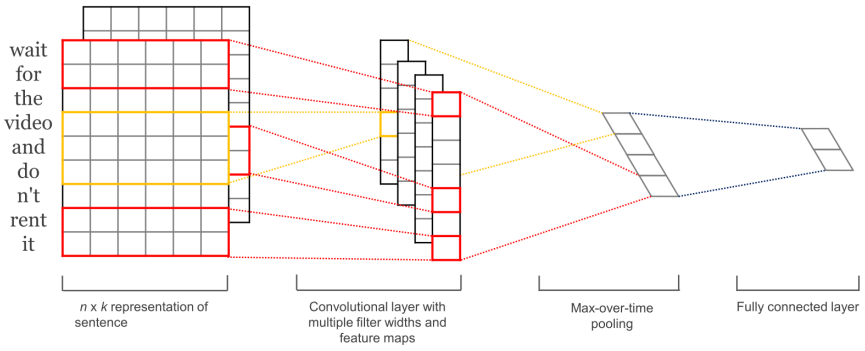


Figure 1: The architecture of the convolutional network [12]

2.7.3 CNN

Pursuing on our considerations on the strucutre of the text, we also propose a convolutional model. Convolutional architectures are very popular in image analysis tasks, because they can detect local patterns in the input data, and can adapt to different input sizes. Our architecture is inspired from the famous work of Kim et al. [12]. The embeddings are fed into three parallel 2D-convolution layers with kernels of different shapes. Our implementation chose kernels of size 3,4 and 5 along the time axis, and covering the whole embedding (see Figure 1). After that, we apply max pooling along the time axis for each channel, and feed the resulting nodes into a fully connected layer. This architecture has the advantage of being able to adapt to any essay length (thanks to the pooling), and should be able to detect syntactical richness thanks to the convolutional layers that shed light onto the local structure of the input.

3 Experiments

We conduct a thorough analysis to compare and discuss all the proposed learning strategies.

3.1 Validation metrics

We train our models using the Mean Squared Error (MSE) loss which is a commonly used loss for regression. It is the sum of the squared distances between our target variable (scores given by examinators) and predicted values (scores outputted by our neural network). During the training process the weights of the neural network are adapted so as to minimize this loss.

Grading methods being different for each essay set, the MSE loss is not the most accurate metric to assess the performance of the model. For this reason it was decided during the Kaggle competition to compare submissions according to another validation metric: the quadratic weighted kappa. This metric captures the inter-rater reliability for qualitative items which are, in our case, the scores given by the true examiners and our AI examiners. To be able to use this metric, scores have to be reshaped to their original discrete values. That is because this metric take as input categorical items. The robustness of the kappa statistic comes down to its ability to account the possibility of the agreement occurring by chance. The quadratic weighted kappa is defined by:

$$\kappa = 1 - \frac{\langle W, O \rangle}{\langle W, E \rangle} \tag{2}$$

It this equation $\langle \cdot, \cdot \rangle$ represents the scalar product between matrices. For scores ranging from $1, 2.., N$, the weight matrix is defined by:

$$W = (W_{i,j})_{(i,j)\in[\![1,N]\!]^2} = \left(\frac{(i-j)^2}{(N-1)^2} \right)_{(i,j)\in[\![1,N]\!]^2} \tag{3}$$

$O_{i,j}$ is the number of essays that were rated i by the true examiner and rating j by the AI examiner. E is an N-by-N histogram matrix of expected ratings (the outer product of each examiner's histogram of ratings). It is normalized so that E and O have the same sum. The quadratic weighted kappa ranges from -1 to 1 where 1 is reached when ratings for both examiners match perfectly. Values above 0.6 are considered to be really good scores (according to Kaggle competition guidelines).

3.2 Experimental protocol

To compare all the configurations we use 5-fold cross-validation on the training data (for each fold the training data is split into subtraining and subvalidation sets) The best epoch is found by looking at the lowest loss value on the subvalidation set. We save the average validation metrics across all folds corresponding to the best epoch for each fold. When all configurations have been cross-validated we test the best configuration on the test data (which remained unseen up to this point).

3.3 Results

All the results for the experiments are saved as things progress in an excel spreadsheet. The raw file containing all these results can be found at

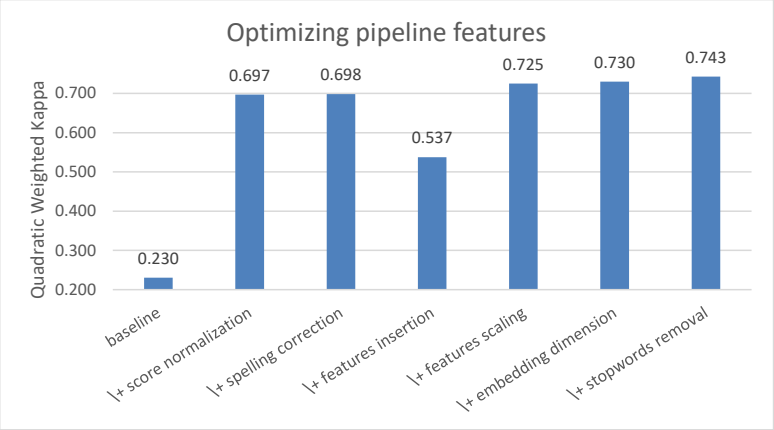


Figure 2: Pipeline features incremental optimization

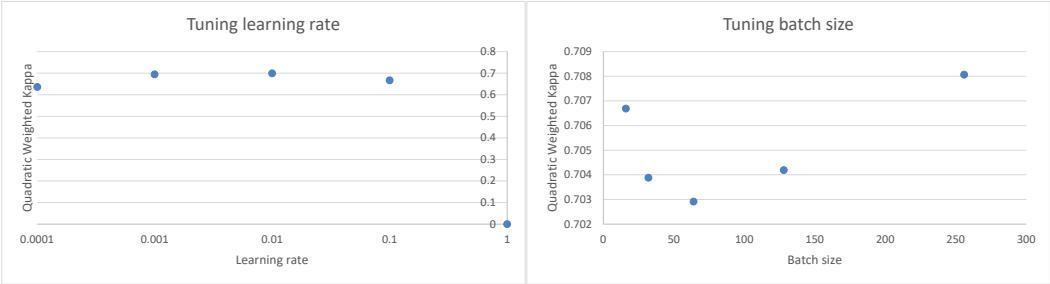
`doc/raw_results.xlsx` in the Github repository coming alongside with this report.

3.3.1 Model selection

In this first part, we describe the process we followed to find our best-performing model.

Pipeline optimization We show on Figure 2 incremental results of pipeline features optimization compared to a baseline method. The baseline method corresponds to the set of default arguments (Word2Vec embedding of dim 50, shallow fully-connected model without dropout, no extra features, no special data preprocessing, learning from all essay sets at once). We found that the baseline method was rather unstable (sometimes giving decent results, sometimes not converging at all). Normalizing score so that they fit in $[0, 1]$ enabled a great improvement of the quadratic weighted kappa metric compared to the baseline method. Spelling correction led to maginally better scores. Extra-features insertion led to more instabilities but adding this together with feature scaling solved this issue and further improved the results. Changing the embedding dimension from 50 to 300 and removing stopwords helped improving the validation metrics even more.

Hyperparameters tuning We then focused on tuning the key learning hyperparameters. As it can be seen on Figure 3, two choices for the learning



(a) Learning rate tuning (b) Batch size tuning

Figure 3: Hyperparameters tuning

rate (0.001 and 0.01) led to similar results. We selected 0.01 because it was slightly better. For the batch size, results indicated that we should select either a small batch size or a very large one. As it speeds up the learning process we decided to go with a rather large batch size (256).

Dense model optimization We tried various architectures for the fully connected model. We show on Figure 4 the results for different dropout values and number of parameters. The results shown correspond to learning from embedded essays as well as extra-features. We also tried learning solely from extra-features a reached a quadratic weighted kappa value of 0.678 which is far from the best results we can get when combining embedded essays and extra-features. The best configuration for the dense model is obtained with hidden layers of size 300 and 128 and dropout of 0.2.

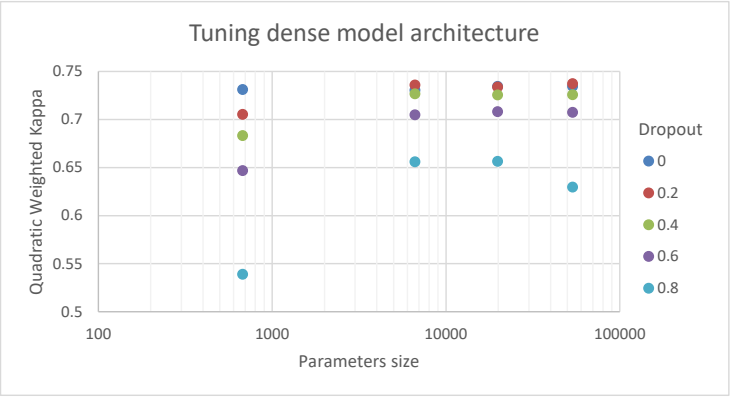
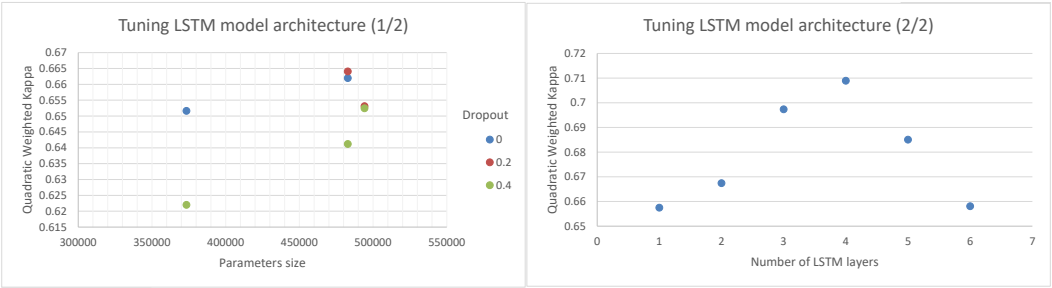


Figure 4: Dense model architecture parameters tuning



(a) Tuning the LSTM's hidden layer size (b) Tuning the LSTM's number of layers

Figure 5: Tuning the LSTM's hyperparameters

LSTM model optimization We conducted similar experiments for our LSTM model, and the results are presented on Figure 5. The best LSTM model was obtained with hidden layer size of 100 for the recurrent units and 16 for the fully connected part and droupout value of 0.2. We then tried deeper model architectures by stacking recurrent units on top of each other. Results show that a depth of 4 is optimal in our case.

CNN model optimization As shown on Figure 6, we tried tuning both dropout and the number of channels for the CNN. Indeed, a 0 dropout seemed to result in quick overfitting, and increasing dropout helped reduce it. The optimal value for dropout for our model is 0.2, but even with this, this model is the least performing: on the validation set, the weighted kappa tops at 0.62, whereas dense models can achieve 0.74. More particularly, the CNN is outperformed in all the essay sets by other architectures. The way we interpret this result is that local structure is not predominant in the grading of the essays, and a larger essay-wise view is very beneficial.

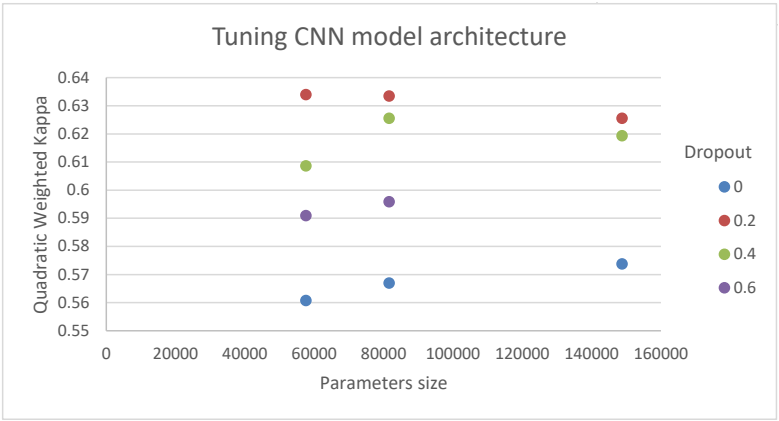


Figure 6: CNN hyperparameters tuning

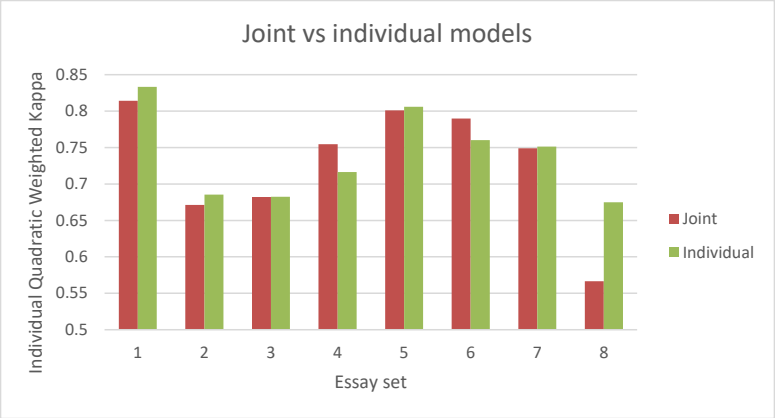


Figure 7: Individual performances for each set on the validation and test sets.

Joint vs individual models Multitask models - or joint models - are increasingly popular because achieving good performance for one task can help to tackle other tasks. However, when we compare the results of our joint model to the ones of individual models we achieve similar performance. Indeed, Figure 7 shows that the quadratic weighted kappa for the joint model is 0.741, and only rises to 0.744 when compiling results together from individual models. The small difference is due to the joint model not being very good on the 8^{th} essay set due to size imbalance between sets. Nevertheless, we prefer to keep the joint model as it is much faster to train and less likely to overfit.

Word embeddings comparison We compare all the proposed word embedding strategies on Figure 8. It turns out that the most successful word embedding strategy is the Word2Vec embedding trained on the essay sentences. We think this is due to specific words and contexts being good indicators for the value of an essay. This strategy leverages the whole corpus thus having task specific knowledge for each of the essay tasks.

3.3.2 Best model performance analysis

The previous section shows that our best-performing model is the dense architecture with the extra-features combined with a word2vec embedding and some dropout. Now, we evaluate the performance of this model on the test data - which has not been used until this point -, compare it with the competition's results, and provide a brief heuristical analysis about the keywords detected by our model.

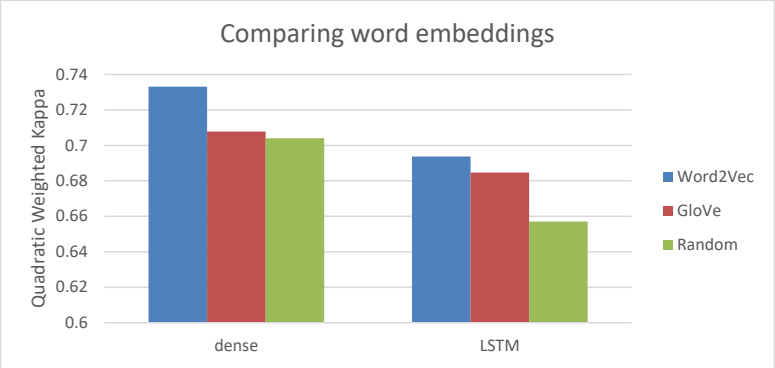


Figure 8: Word embeddings comparison

Test results We show the performance of the best model on the test set compared to the performance on the validation set on Figure 9. The individual quadratic weighted kappa are equivalent for all sets except for the 8^{th} set. The reason for this is that the score range is the largest for the 8^{th} set and the kappa metric is expecting identical scores to count true positives. So even if the scores are close the kappa can be low. Moreover the 8^{th} set contains less training samples so it is harder to generalize to unseen essays for this set. All in all, the overall performance is quite satisfying and comparable to other works dedicated to this Kaggle competition.

Words sensitivity analysis We propose here a simple idea to analyse the sensitivity of our model to particular keywords. We recall from Section 2.7.1 that the dense model starts by averaging all of the embeddings of the essay, and takes this mean embedding as the input of the dense layers. Hence, it is very easy to feed only one word to the model. Heuristically, this corresponds to automatically grading the essay composed of this sole word. By doing that for all words in the vocabulary, we can map a grade to each word, and analyze this grade as the importance given by the algorithm to this word.

Caveat - a word on extra features This simple reasoning is only valid if the only input to the model is the essay's mean embedding. As a matter of fact, our model also requires the extra features described in Section 2.6. To make the model operate near its training zone and have meaningful results, we do the following : for each set, set each extra feature to its mean value across the set, and feed the embedding of the word as the input. The output of this algorithm is, for each word, a set of grades.

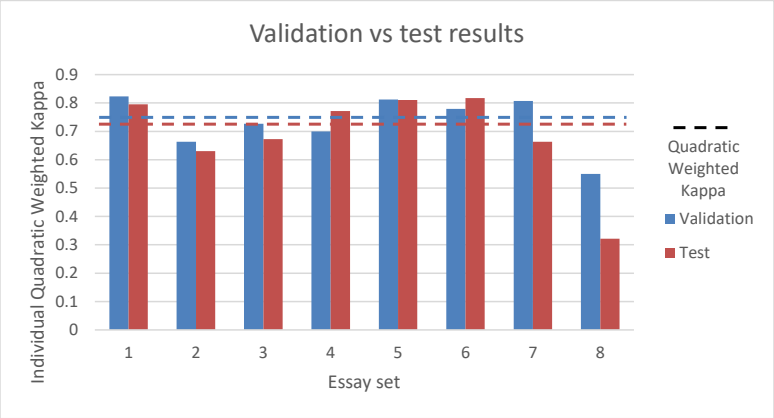
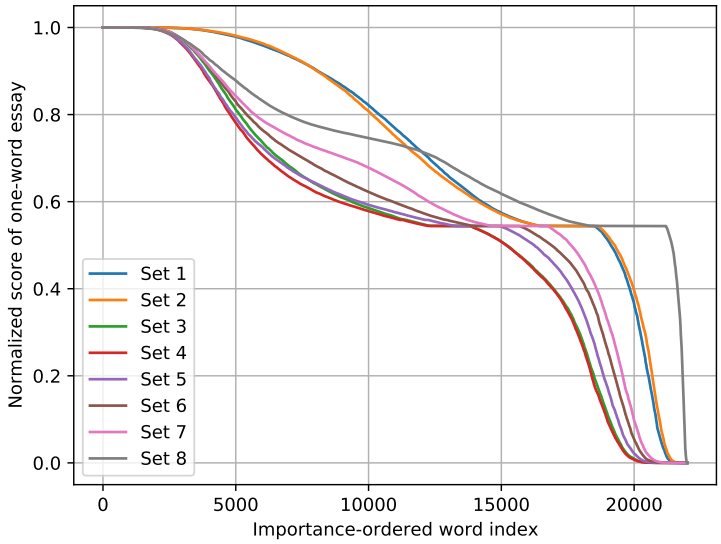


Figure 9: Comparison between the results on the validation and test sets.

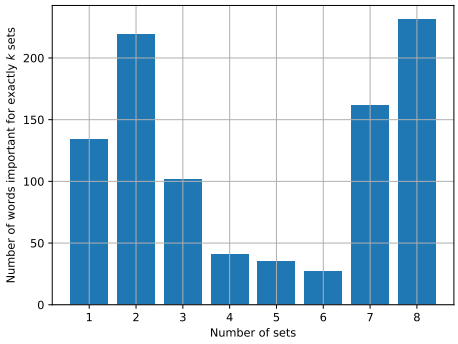
However, it is now *wrong* to interpret these grades as "the grade given by the algorithm for essay set number k to the essay consisting of the sole keyword". We will keep that in mind, but neglect it in the analysis.

Grades of one-word essays The results of this algorithm are shown in Figure 10. Very interestingly, one can see that it is possible to totally fool the algorithm with essays composed of only one particularly well-chosen word. While being unsurprising, this result shows that the algorithm is not very robust and is very susceptible to adversarial attacks, and hence not suited to be used in real-world applications.
We can further investigate this figure by noting that the shape of the importance profile of Figure 10a does not seem to be correlated with performance. Indeed, Sets 1 and 2 have very similar curves, yet the model performs very differently on these two tasks (see Figure 7).
Figure 10c shows that, interestingly, the model is able to identify more words that matter only for these topics for Sets 1, 2 and 8 : excluding set 7, these are the sets corresponding to argumentative tasks (Table 1). Once again, this is not correlated with performance, as it can be seen on Figure 7.

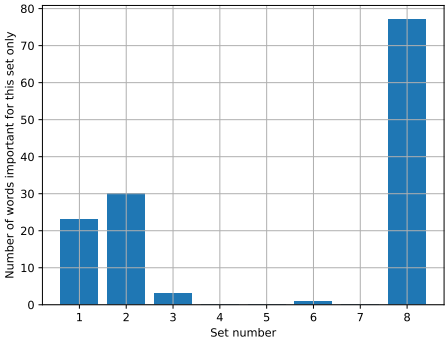
Comparing topics and important words You will find in Annex A an overview of the important words for sets 1,2 and 8. Surprisingly, most of these words are not in the lexical field of the topic upon wich the essay is about, although there are some exceptions. What's more, some words



(a) Sorted words importance.



(b) Words with importance of 1.0 in exactly k sets.



(c) Words with importance of 1.0 for set k only.

Figure 10: Results of the importance algorithm. Figure 10a is plotted after sorting the importances independently for each set, so the word corresponding to an index depends on the set. Figures 10b and 10c give information on the correlations between the important words across sets.

marked as important have spelling mistakes[6], such as "teconogly" instead of "technology" for Set 1, "netflicks" instead of "netflix", or even abbreviations or slang talk such as "bf" instead of "boyfriend", or "ganna" that probably stands for "going to". This is probably accounted for by the fact that the dataset is relatively small for deep learning tasks, and more training examples would probably reduce this phenomenon.

4 Conclusion

We have proposed in this project a neural-network based approach to the automated essay scoring task, and have tried different architectures combined with a large variety of pre-processing options and different embedding techniques. The best results were obtained by using a fully-connected network with a Word2Vec embedding, to which we fed the essay itself and a few global features we computed beforehand on the whole essay. This approach got a kappa-score of about 0.74, which is comparable to the leaderboard of the original Kaggle competition.

We have started this report by stating how much of a game changer a reliable automated scoring system could be. This project has shown us how difficult designing such a system can be, even with the wonderful toolbox offered by modern natural language processing techniques. By testing several algorithms, we ranked 23$^{\text{rd}}$ on the competition's leaderboard[7], yet our algorithm could be easily tricked into giving a high grade to nonsensical essays. It is clear that the basic architecture of the model as well as the small size of the dataset are at fault here, since we cannot reasonably hope for our model to learn grammar. This shows how large datasets and thorough modelization are essential for a model to capture the whole complexity of an essay. However, gathering more data for such a task is very costly. Indeed, contrary to translation tasks where huge databases can be constituted using existing texts, tests on a particular topic are generally taken only by a handful of students (it is not common for more than a few thousand students to work on the same test), and the resulting essays are generally handwritten and not adapted to being processed by computers. Finally, it requires a lot of human work to rate an essay, much more than to label an image for instance. Hence, even with a better architecture and more computer power, the constitution of the dataset will still be a problem.

In spite of all that, this project has helped us shed light on some of the

[6] Although this could be accounted for by our spelling error correction algorithm

[7] By extrapolating the results obtained on our own test set to the competition's true one, to which we don't have access.

criticisms that were made to automated scoring systems in the past years. It is true that learning-based algorithms primarily focus on structural and formal features that do not necessarily reflect the meaning of the sentence. In our case, this was very obvious, but more sophisticated algorithms are not immune to this. However, this does not mean that learning algorithms will never be useful for grading. One can imagine a combination of human rating and automated rating to detect rating errors, or use learning-based algorithms for what they are really good at : detecting structures in data, by checking for instance the grammatical correctness of an essay, or other task that largely rely on these surface features that neural networks can detect.

A Important words

The results presented here can be found in the folder `outputs/000001/processed/` of the GitHub page of our project. The words given here are the words that are assigned a maximal importance (that is, the corresponding one-word essay obtained the highest possible grade) *only for the set considered*. We interpret these as keywords that the algorithm has identified and that are specific to the considered task.

A.1 For set 1

Topic Write a letter to your local newspaper in which you state your opinion on the effects computers have on people.

Words adulterated, appointed, attainable, beards, doesnt, extravagant, freer, gunk, haren, jim, loathe, marble, mooned, nerd, portals, proceeds, shimmered, teconogly, treadmills, twinkle, unpronounceable, unsocial, venting

A.2 For set 2

Topic Write a persuasive essay to a newspaper reflecting your vies on censorship in libraries.

Words anythingelse, athe, carie, ceilings, deflating, durning, eto, frombeing, hardworked, hordes, impressive, moutain, mover, muster, netflicks, overworked, prevails, redon, restated, snooty, solider, sprawled, squires, strats, thistle, tracked, trown, vaccine, watchin, weakly

A.3 For set 8

Topic Tell a true story in which laughter was one element or part.

Words appicatrone, aprons, arjuna, ash, bad, bf, building, bup, calorie, cashews, cheeses, comit, contradiction, crazy, dace, detected, exaggerate, extravaganza, figer, funny, ganna, get, girls, go, guy, guys, heartfilling, history, hogweed, hopper, interviewing, italy, jeopardize, kids, knew, know, let, looked, looking, mauerie, men, might, mom, monstrous, never, older, omg, oppernutity, patient, pediatrician, peeling, pick, pingpong, programe,

really, rebuilt, room, satisfies, shaked, shead, shrouded, slur, sometimes, stables, started, store, stray, stuff, suspenseful, tarts, turned, unicycles, unlocks, want, watched, waver, wo

References

[1] "The hewlett foundation: Automated essay scoring." https://www.kaggle.com/c/asap-aes/. Accessed on April 30th.

[2] L. Iribarnegaray, "« c'est le concours de notre vie, et ce n'est plus du tout égalitaire » : le défi des candidats aux études de santé," *Le Monde*, 2020.

[3] M. Winerip, "Facing a robo-grader? just keep obfuscating mellifluously." https://www.nytimes.com/2012/04/23/education/robo-readers-used-to-grade-test-essays.html, The New York Times, 2012. Last visit on May 2nd, 2020.

[4] H. Nguyen and L. Dery, "Neural networks for automated essay grading," 2018.

[5] S. Song and J. Zhao, "Automated essay scoring using machine learning," *Stanford University*, 2013.

[6] J. Nicolich, "Automatic student essay assessment." https://github.com/Turanga1/Automated-Essay-Scoring/blob/master/Automatic%20Student%20Essay%20Assessment.pdf. Last visited on May 2nd, 2020.

[7] A. A. Manvi Mahana, Mishel Johns, "Automatic essay grading using machine learning." https://github.com/m-chanakya/AutoEssayGrading/blob/master/papers/paper1.pdf. Last visited on May 2nd, 2020.

[8] E. B. Page, "The imminence of... grading essays by computer," *The Phi Delta Kappan*, vol. 47, no. 5, pp. 238–243, 1966.

[9] Y. Attali and J. Burstein, "Automated essay scoring with e-rater® v. 2.0," *ETS Research Report Series*, vol. 2004, no. 2, pp. i–21, 2004.

[10] L. M. Rudner and T. Liang, "Automated essay scoring using bayes' theorem," *The Journal of Technology, Learning and Assessment*, vol. 1, no. 2, 2002.

[11] K. W. Murray and N. Orii, "Automatic essay scoring," *IEICE Transactions on Information and Systems*, 2012.

[12] Y. Kim, "Convolutional neural networks for sentence classification," *arXiv preprint arXiv:1408.5882*, 2014.

[13] K. Taghipour and H. T. Ng, "A neural approach to automated essay scoring," in *Proceedings of the 2016 conference on empirical methods in natural language processing*, pp. 1882–1891, 2016.

[14] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[15] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1532–1543, Association for Computational Linguistics, Oct. 2014.