SupER: A Supervised Entity Ranker for the Triple Scoring WSDM 2017 Task

Jason Wong 404610477[1], Vansh Gandhi 204621797[1], and Daniel Kim 204446633[2]

[1]Department of Computer Science, University of California, Los Angeles
[2]Department of Mathematics, University of California, Los Angeles

***Abstract*— When querying a knowledge base, it is useful to rank the resulting list of entities in terms of their relevance; the Triple Scoring WSDM 2017 task is centered around this challenge. Given a tuple that specifies a person and a descriptor, which can be a nationality or a profession, we are to assign a score that represents the relevance of the tuple, compared to the other tuples of the same relation. We have developed SupER (Supervised Entity Ranker), which uses supervised learning through a recurrent neural network, to solve this problem. SupER extracts features from the Wikipedia page for each tuples person. Our feature extraction algorithm assumes that frequent occurrences of the descriptor phrase in certain structures of the Wikipedia page, such as the infobox and the introduction section indicate that the descriptor is relevant to the person. Further, the feature extraction algorithm makes the assumption that occurences of the descriptor are not equal in suggestive power; an occurence of the descriptor that appears in the topmost infobox is more significant than an occurence of the descriptor that appears in the notes and references at the bottom of the page. The model is trained on these features using ground truth scores provided by the problem specification, and produces accuracies of approximately 0.70 and 0.68, when tested with validation profession tuples and nationality tuples, respectively.**

I. Introduction and Background

Given a tuple from a type-like relation, one is asked to implement a score from a discrete scale from 0 to 7 that assesses the relevance of the statement expressed by the triple relative to other tuples from the same relation. Such a score has proven to be useful in the field of information retrieval; with the large amount of data made available in knowledge bases such as Wikipedia, it is often of interest to rank groups of data by their relevance. The task, inspired by the Task Chairs' publication, Relevance Scores for Triples from Type-Like Relations [4], calls upon us to focus on two attributes: profession and nationality.

The structure of this assessment is defined as follows:

(name, descriptor, score)

where name corresponds to the name of the person, relation is one of the two attributes, and score measures how close the person is to the descriptor. Here are two examples for the profession relation:

A. R. Rahman Entrepreneur 0
Dr. Dre Songwriter 4

The organizers provided the annotations of 385,426 persons in which only people from the already provided list were used in the test sets. The participating systems were then evaluated against two test sets: one for professions and one for nationalities.

One key note of observation is that the assignment put no limits in the kind or amount of additional external training data that could be used by the participating systems. For each tuple, seven judges were asked to make a binary decision. Afterwards, the sum of all positive scores yielded a score from 0 to 7. This means the results were not fact-based, and were not quite objective.

II. Problem definition and formalization

Given a file containing a list of tuples in the format, (person, descriptor), where descriptor can be a nationality or a profession, we must assign a score in [0,7], which indicates the relevance of the tuple, in comparison to the other tuples in the same relation. Our output file will be exactly identical to the input file, with an additional column appended, containing the score our algorithm has assigned for each row.

Each person provided in the input file can be found from the persons file included in the problem specification. The person names are identical to the names used by the English online Wikipedia; to access the Wikipedia page for a given person, we may navigate to http://en.wikipedia.org/wiki/**person_name**. Each descriptor is taken from the list of 200 professions found in the professions file, or from the list of 100 nationalities in the nationalities file.

The profession.train and nationality.train contain ground truth values, and the profession.test and nationality.test files contain the ground truth values used to test the final submissions of the task.

The script, evaluator.py is used to evaluate the performance of our submission, in terms of accuracy, average score difference, and Kendall's Tau.

While not explicitly stated in the problem specification, it is assumed that, as seen in the training and testing files, tuples for a particular person will appear in one continuous sequence in the file, and will not appear elsewhere in the file.

III. Methods

System requirements: Our software is to be run in a bash shell, with python3 [7] installed. The Python modules we require are Keras v2.1.5 [6], Numpy v1.14.1 [9], and TensorFlow v1.6.0 [1].

Because we decided to use a machine learning algorithm, we realized that we needed to convert our inputs, all of which are textual in nature, into some sort of numerical input. We considered using the Word2Vec model in order to represent this data. We also reviewed the Task Chairs' publication, Semantic Search on Text and Knowledge Bases [5] and considered applying the the strategies discussed in their survey of semantic search on text. However, these approaches would add additional computational cost, and would attempt to gain a higher understanding of the context of entities than was necessary. So,

we first decided to make our neural network independent of the actual person who the query was for. In order to do this, the network we created would need to be able to take as input a feature that was person agnostic. We achieved this by running a preprocessing algorithm that converted data into a standard format for each person, and feeding the appropriate preprocessed data as the input to the neural network. This preprocessing was implemented in minefeatures.py, which runs a preprocessing algorithm that, when given the contents of that persons Wikipedia entry, along with the name of a person, and a descriptor phrase, searches the given Wikipedia file for any instance of relation of the descriptor with the corresponding user. With every instance of a descriptor inside lines, the algorithm sets a value corresponding to its importance in the wiki doc.

Therefore, this feature extraction algorithm relies on the assumption that occurrences of the descriptor in certain structures of the Wiki page, are strong indicators of how relevant the descriptor is to the person. Furthermore, it is assumed that some appearances of the descriptor are more powerful than others: in this instance, areas such as the introduction paragraph and the vertical box which lists key features of the person have very high point values, while body paragraphs and references have relatively low values. The feature extracted by this algorithm provides a rough starting point of values for each person that will be used to train the recurrent neural network. The output of the preprocessing algorithm was in the format of 1-dimensional array, where each entry in the array corresponds to a specific descriptor (nationality or profession). This is possible because we are given the set of 100 nationalities and 200 professions that could be used as the descriptor; this preprocessing stage mines the point values for each of these possible descriptors, for each of the possible 385,426 people names provided in the persons file provided with the problem definition.

One major obstacle that came with compiling a suitable initial score was processing specialized attributes or dealing with compound words. For example, person X might be labeled as a physicist for his profession, but that word might have little to no matches in the actual Wikipedia file but instead may have an abundance of words that are related to physic(s). A similar problem occurs with compound words. In order to more accurately assess a score, the algorithm runs a recursive step if the Wikipedia file fails to find matches or encounters a space or - character in the keyword. This recursive step runs a portion of the keyword so that it would be more likely to find results within the Wikipedia document. The combined score of the main and the recursive step are weighted by a constant and then averaged to create a more accurate score.

One obstacle that we could not overcome with this implementation was an over approximation of keywords with increasingly shortened size. An example would be how actor shortened would become the keyword act but would match with incorrect words like react. Our solution was to create a distance function that correctly displaces a keyword to the first position of a viable match, thus ensuring that points would be tallied correctly. The problem with implementing such a feature is that it would dras-

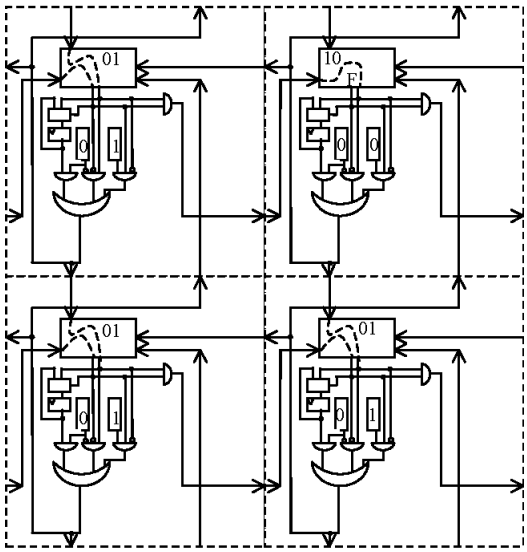


Fig. 1. A diagram that overviews the structure of a standard one-hot array. The diagram emphasizes the efficiency of the boolean structure when sorting data [2].

tically increase runtime because all possible matches would have to be searched through an additional loop for a matching space character. Additionally, HTML sentences very consistently end characters with symbols instead of characters we are trying to find, therefore decreasing the actual accuracy of the assessment. For these reasons, we weighed the cost-benefit of this implementation and decided to avoid it for optimization.

In order to actually score a tuple, we made use of a neural network machine learning algorithm, which takes as input the preprocessed wikipedia data of the person in question, as well as the one hot array of the descriptor we are scoring for. Given these inputs, the neural network then spits out a 1-dimensional, 8-element array, where each item in the array represents the probability of that score being true for the (person, descriptor) tuple. We then post-process this output array to convert it into a one-hot array. The process of the converting it to a one-hot array is to assign the value 1 to the index containing the element with the highest value. All other elements are set to 0. Then, the index at which the value 1 occurs is the score for that tuple. The benefits of using a one-hot array is an increased memory efficiency that results from using boolean values instead of the standard Int value, thereby increasing model calculation times.

The neural network model we used is a Sequential model. This means that it applies a selection of layers one by one, one after another. Within this model, we used 9 layers, of which 7 were hidden layers, 1 was the input layer, and 1 was the output layer. All layers used were Dense layers, with the exception of 2 layers. We used a Flatten layer in order to reshape the multi-dimensional input into a shape that was more easy going for computation and comprehensibility. The Dropout layer, at a rate of 50 percent, was used in order to avoid overfitting, as it sets half of the neurons in the layer to use zeroes as their input, thereby making sure to introduce some noise into the data so that the network learns an overall generalization of the data rather than overfitting to the training data set.

In order to train the network, we made use of the example data provided to us to use as the training data set. Running the command
python train.py
initiates the training process, in which the Python program grabs the training data from a specific training file, builds the appropriate model, and then fits the input and output data to that model. We made

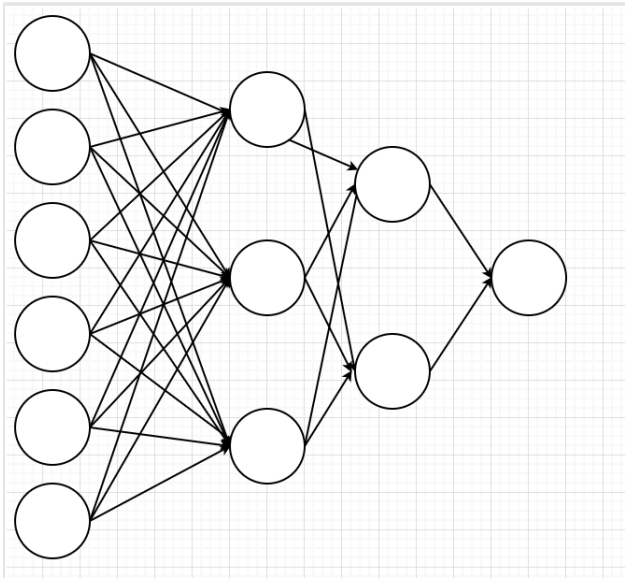


Fig. 2. A high level representation of a neural network. The model used for this project has many more neurons, making it difficult to accurately depict.

the decision to run the fitting for 200 epochs. This was the iteration at which we found a good tradeoff between overfitting the data, underfitting the data, runtime, and accuracy (which started to plateau, further indicating that 200 epochs was a good place to stop). Once the model is trained, we save the trained model into its own file, so that it can be easily recreated when trying to score real data without having to completely retrain. The profession model and nationality model are saved into 'professions.model.hd5' and 'nationalities.model.hd5', respectively.

Once the model has been created, trained, and stored, the 'scorer' can successfully begin to run on an arbitrary data set. The scorer first loads the model saved by the trainer. Then, for each name in the input, it fetches the preprocessed wikipedia array data for that person. It sends that, along with the descriptor for the person, to the model to predict what the score should be. This score is then output as the next column to the output file.

The main entrance to the scorer, once the model has been adequately trained is by running

python main.py -i <INPUT_FILE> -o <OUTPUT_DIRECTORY>.

The ./run.sh script will automatically run all the aforementioned scripts, in the correct order with the correct inputs. That is, it will run train.py for both nationality and profession and save the model to a file. Then it will run the validation data through the model to ensure there is a good accuracy for both profession and nationalities. Then, finally, it will run through the test data and predict the values of the output.

There is also a seperate Makefile, which ./run.sh is actually a proxy for. This Makefile contains various other commands as well, such as the scripts to actually scrape the Wikipedia data, and mine and preprocess it, as well as the training and testing commands for testing, validation, and training data.

IV. Experimental Design and Evaluation

1) Accuracy: the percentage of triples for which the score computed by our system differs from the actual score by at most 2.
2) Average score difference (asd): the sum of the absolute differences of the system-generated scores and the ground truth scores, divided by the total number of triples.
3) Kendall's Tau: for each relation and for each subject, compute the ranking

of all triples with that subject and relation according to the scores computed by the system and the score from the ground truth. The difference of the two rankings is then computed using Kendall's Tau.

One of the first things we had to figure out when implementing our recurrent neural network was how we were going to represent our inputs. We already knew the format of the preprocessed Wikipedia data was an array of numerical values, so that would be fine as is. However, we needed to represent the person's descriptor in the tuple as some sort of numerical value. Once again, we considered using the Word2Vec model, but rather than computationally not making sense, in this scenario, it did not make semantic sense, as there didnt need to be much interpretation of the actual descriptor itself, as the preprocessed data took care of interpreting the descriptors. Therefore, we were able to represent each descriptor as a specific and consistent index that also matched up with how the descriptors were organized in the preprocessed data. This meant that the neural network would be able to recognize patterns between the the index of the descriptor and a value in the array, without needing any semantic meaning.

One challenge we had to overcome was incorporating a way to test our machine learning data using actual test results. For this challenge we incorporated a highly supported testing schematic entitled 80/20 test validation. Through this systematic testing scheme, one uses only 80 percent of the data results to organize a highly accurate model that will be used to test the neural network on the final 20 percent for actual accuracy of the created model. Because experts in the field of machine learning emphasize the optimization of this split, our model for testing used similar principles.

Another challenge we experienced was how to correctly represent the score that is outputted from the recurrent neural network. Initially, we represented the score as a single numerical value. During training, scores would always fall into the range [0-7], but the outputs of the recurrent neural network did not always fall into this range; sometimes, the scores would be as small as -1, and as large as 8. Furthermore, the scores would also be floating point numbers. This would cause our output to fail the sanity checks performed by the evaluator script. Our initial solution to this was to write the exact scores produced by the recurrent neural network into the output file. When all tuples have been scored, loop through the whole file again, keeping track of the max and min scores. We would use these maximum and minimum scores as the bounds for a linear mapping to our desired range. That is, the minimum score would map to 0, and the maximum sore would map to 7, while all the numbers in between would be guaranteed to map to a number within [0,7]. Finally, we rounded non-integer scores to the closest integer, and re-wrote the output file. This did allow our outputs to pass the sanity checking performed by the evaluator script. This model produced an accuracy of 0.53, an average score difference of 2.34, and an average Kendall's tau of 0.45

In the next iteration, we represented the score in a one-hot output array of length 8; to represent the score n within the range

[0,7] the output array at index n is set to 1, while all other entries are set to 0. This adjustment resulted in outputs that easily mappable to an integer score; there would be one strong entry in the one-hot array that was a decimal value near one, and the rest of the entries would be near 0. To extract a score from this one-hot formatted array, we simply return the index of the maximum entry. This not only simplified the code to produce our output score, but also resulted in improved performance and accuracy; without any additional modifications to the model except for this one-hot encoding scheme of the output, this model produced an accuracy of 0.58, an average score difference of 2.31, and an average Kendall's tau of 0.41, showing that this change did result in some performance improvement.

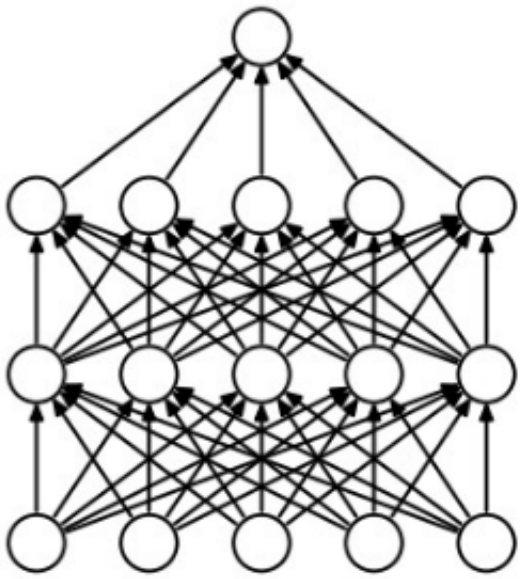
Only once we had decided what the format and structure the inputs and outputs were going to be represented in could we accurately begin to optimize the model for our neural network. In order to properly build our model, we had to consider a variety of factors, including number of layers, number of neurons in each layer, the activation function for each layer, how to avoid overfitting, whether to introduce noise into the data, whether to reshape the data, which loss function to use, and what criteria determined an appropriate number of epochs. The first thing we knew was that we wanted to avoid overfitting. As previously mentioned, we used Dropout to avoid overfitting. According to Peirre Baldi and Peter Sadowski, in their paper Understanding Dropout [3], the most optimal rate is 0.5, or 50 percent, which was proved with various simulations to produce the best results.

We used the ReLU as the activation function for all but one of our layers because we knew we did not want to activate on inputs less than 0, and wanted low outputs on values around 0 as well. We decided to use tanh for one of our layers as it has a steeper slope, which more firmly discouraged positive outputs for low occurences of a descriptor, and encourages it for high occurrences. The determination of the optimizer function was mostly trial and error, once we had all the other parameters set. We found that the RMSProp optimizer did significantly better than the Stochastic Gradient Descent in getting a higher accuracy. We used the default loss function of mean squared error.

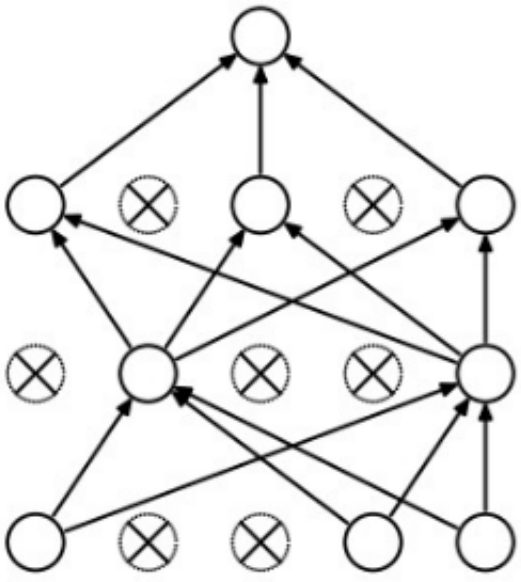
V. Related Work

The following table corresponds to the scores for the actual WSDM competition for Triple Scoring. For our results we were able to reproduce accuracy scores ranging from .65 to .75, and a Kendalls Tau ranging from .25 to .35. We were unfortunately unable to reproduce results that matched the higher levels of accuracy with the likes of bokchoy, cress, or goosefoot, but because we were unable to access or implement methods with their code, most of the work was done using our own ideas and implementations. As a result, we were pretty satisfied with the results we were able to produce on this challenge.

The submission that most closesly mirrored the solution we came up with was the Lettuce project. Lettuce also picked machine learning with neural networks as their approach of choice. However, taking a look at the models they constructed, it



(a) Standard Neural Net



(b) After applying dropout.

Fig. 3. With dropout, some of the neurons in the network are essentially ignored, with around a 50 percent probability. This prevents overfitting the data [8].

Team	Accuracy	Avg. score diff.	Tau
goosefoot	0.75	1.78	**0.31**
cress	0.78	**1.61**	0.32
bokchoy	**0.87**	1.63	0.33
chaya	0.70	1.81	0.34
cabbage	0.74	1.74	0.35
chicory	0.63	1.97	0.35
kale	0.69	1.85	0.36
lettuce	0.82	1.76	0.36
gailan	0.70	1.84	0.37
chickweed	0.77	1.87	0.39
fiddlehead	0.73	1.70	0.40
radicchio	0.80	1.69	0.40
bologi	0.68	1.91	0.41
catsear	0.80	1.86	0.41
cauliflower	0.75	1.87	0.43
samphire	0.78	1.88	0.44
celosia	0.69	1.93	0.45
rapini	0.73	2.03	0.45
yarrow	0.60	2.04	0.45
endive	0.55	2.49	0.46
pigweed	0.74	1.94	0.48

Fig. 4. A diagram of the results of the WSDM 2017 Triple Scoring Competition. [10]

is apparent that they used more advanced techniques. Lettuce extracts many of the semantic meanings of words, and also considers synonyms and similarly structured words and phrases as the same entity. Therefore, their algorithm does not rely upon the Wikipedia data containg the exact descriptor in question, but rather extracting the definition of the descriptor from the Wikipedia data.

Furthermore, the Lettuce project analyzed the meaning of whole sentences rather than individual words, which means they were able to learn even further based on the context in which the word was used in rather than just blindly taking notice that a word appears and counting it.

The top scoring project, Bokchoy, made use of a variety of classification techniques, implemented within the scikit-learn package, such as TfidfVectorizer, CountVectorizer, RandomForestClassifier, and LogisticRegression. The TfidVectorizer and CountVectorizer combination is notable be-

cause we attempted to implement a similar method in the preprocessing of our data, where we essentially created a vocabulary based on the provided persons and descriptors, and counted relevant occurences within Wikipedia documents. However, we were not able to achieve a similar score, due to the differences in processing the inputs.

VI. Conclusion

We have presented SupER, a software package that uses supervised learning through a recurrent neural network, to score entities for the WSDM 2017 Triple Scoring Task. SupER relies on features mined from Wikipedia. We mine features for nationalities and professions, under the assumption that the appearance of descriptor phrases in different sections of the Wikipedia page indicate relevance of the descriptor to the person. SupER produced 0.70 and 0.68 when scoring nationalities, and professions respectively, with the final model on validation data. We assumed, as participants, that we did not have access to the testing data, so we did not optimize on it, and only optimized on the validation data. However, the run.sh script will score the testing data with our model.

Future developments include refinements to the feature extraction algorithm, and adjustments to the recurrent neural network model. One weakness of the feature extraction algorithm is the inability to detect similar words. While our edit-distance strategy did not benefit the feature extraction process, we did implement a basic solution to detect similar words by recursively searching for relevant substrings of the descriptor phrase. Still, it would be beneficial to utilize Natural-language processing to find semantic similarity between words. Based on our research on the tools available in this field, we would be interested in experimenting with WordSpace or word2Vec to improve the feature extraction algorithms ability to detect semantic similarities between words.

To improve the recurrent neural network model, the biggest step that can be taken is to train the neural network on a significantly larger dataset. We had around 500 tuples to begin with, of which 80 percent could be used to train. That left us with around 400 training data points. However, to get an accurate model, we would need a couple thousand data points instead. This increased variance in the data would further prevent overfitting and would also allow us to tweak the parameters of the model so as to to allow a more accurate model.

Another possible way to improve the current neural network model is to find an even more efficient learning structure for our given data. While our current learning model covers extensive and specialized methods that boosted our resulting score by a significant amount, a better understanding of machine learning models could have aided us in choosing an even more optimal model specialized for our given assignment.

VII. Task and Workload distribution

Jason Wong created and managed the project's overall structure; this involved writing all the bash scripts to link up each part of the software, along with the main methods to call the required python functions in sequence. He created the en-

try python script that accepts user arguments, and wrote the entry script run.sh that replicates our results. He developed the testing methodology, which involved randomly generating an 80/20 split of the test data, and wrote the Makefile targets which train, validate, and test the model. He developed the scripts to mine the Wiki data in manageable batches that could be submitted as long running background processes. He contributed to the feature mining algorithm, and helped with testing the performance of the neural network model, as the group iterated towards better performance. Finally, he wrote the documentation provided in the scripts/README.txt and data/README.txt.

Daniel Kim was responsible for creating the preprocessing files that provided an initial score representation for the list of tuples made available through the assignment. He participated in mining a part of the 385,000 people provided within the zip file as well as help document multiple sections of this report.

Vansh Gandhi worked on creating the neural network model and working directly with the Keras interface. He iterated upon the neural network model in order to achieve a high accuracy and determined, through an analysis of previous works, what the best parameters for the network should be in order to properly train on the test data and output the scores.

All participants were equally involved in discussing ideas, process, and feedback. All participants helped to mine and preprocess the Wikipedia data. All participants were involved in writing the paper.

VIII. ACKNOWLEDGMENTS

We would like to thank Professor Wei Wang, and the Teaching Assistants, Jyun-Yu Jiang, and Yichao Zhou, for their assistance with this project.

REFERENCES

[1] ABADI, M., BARHAM, P., CHEN, J., CHEN, Z., DAVIS, A., DEAN, J., DEVIN, M., GHEMAWAT, S., IRVING, G., ISARD, M., KUDLUR, M., LEVENBERG, J., MONGA, R., MOORE, S., MURRAY, D. G., STEINER, B., TUCKER, P., VASUDEVAN, V., WARDEN, P., WICKE, M., YU, Y., AND ZHENG, X. TensorFlow: A system for large-scale machine learning, 2016.

[2] ARNOLD, M. G., VOUZIS, P., AND CHO, J. H. Bitstream Efficiency of Field Programmable One-Hot Arrays. In *2010 IEEE Computer Society Annual Symposium on VLSI* (7 2010), IEEE, pp. 436–441.

[3] BALDI, P., AND SADOWSKI, P. Understanding Dropout.

[4] BAST, H., BUCHHOLD, B., AND HAUSSMANN, E. Relevance Scores for Triples from Type-Like Relations.

[5] BAST, H., BUCHHOLD, B., AND HAUSSMANN, E. Semantic Search on Text and Knowledge Bases. *Foundations and Trends R in Information Retrieval Bast, B. Buchhold, E. Haussmann 10*, 3 (2016), 119–271.

[6] CHOLLET, F., ET AL. Keras. `https://github.com/keras-team/keras`, 2015.

[7] PYTHON. Python. `http://www.python.org`, 2001.

[8] SRIVASTAVA, N., HINTON, G., KRIZHEVSKY, A., SUTSKEVER, I., AND SALAKHUTDINOV, R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research 15* (2014), 1929–1958.

[9] TRAVIS E, O. Numpy: A guide to numpy. `http://www.numpy.org`, 2006.

[10] ZMIYCHAROV, V., ALEXANDROV, D., NAKOV, P., KOYCHEV, I., AND KIPROV, Y. Finding People's Professions and Nationalities Using Distant Supervision The FMI@SU goosefoot team at the WSDM Cup 2017 Triple Scoring Task.