ASTES

Advances in Science, Technology and Engineering Systems Journal
Vol. 5, No. 2, 13-18 (2020)
www.astesj.com
Special Issue on Advancement in Engineering and Computer Science

ASTES Journal
ISSN: 2415-6698

Productify News Article Classification Model with Sagemaker

Johannes Lindén[*], Xutao Wang, Stefan Forsström, Tingting Zhang

Mid. Sweden University, Department of Information Systems and Technology (IST), 851 70, Sweden

ARTICLE INFO

Article history:
Received: 29 May, 2019
Accepted: 09 July, 2019
Online: 09 March, 2020

Keywords:
Machine learning
Data mining
Big data
News events
Journalists
Editors
Text analysis
Natural language processing
NLP
Paragraph vectors

ABSTRACT

News companies have a need to automate and make the process of writing about popular and new events more effective. Current technologies involve robotic programs that fill in values in templates and website listeners that notify editors when changes are made so that the editor can read up on the source change on the actual website. Editors can provide news faster and better if directly provided with abstracts of the external sources and categorical meta-data that supports what the text is about. To make categorical meta-data a reality an auto-categorization model was created and optimized for Swedish articles written by local news journalists. The problem was that it was not scale-able enough to use out of the box. Instead of having this local model that could make good predictions of the text documents, the model is to be deployed in the cloud and an API interface is created. The API can be accessed from the tools where the articles is being written and therefore these services can automatically assign categories to the articles once the journalist is done writing it. To allow scale-ability to several thousands of simultaneously categorized articles and at the same time improving the workflow of deploying new models easier the API is uploaded to Sagemaker where several models are trained and once an improved model is found that model will be used in production in such a way that the system organically adapts to new written articles. An evaluation of Sagemaker API was done and it was concluded that the complexity of this solution was polynomial.

1 Introuction

Modelling data with machine learning algorithms has shown to give promising results in various of areas, for example image processing and robotics. The areas are growing and machine learning becomes more advanced for every day. Natural language processing is one area that is difficult and requires larger and deeper networks to perform. The larger a network is the more computational power is required to compute a prediction. For a system in production the speed and memory usage could be fatal to the incoming requests of the application. This article addresses the difficulties to take such a system from the model produced by the algorithm to a productified version utilized by a product or users in real time. The response time, memory consumption and computational power are of importance when these types of models will be available with minimal waiting- and downtime. This article addresses one way to scale the application to handle all requests from the user base of a media company distributed over 22 regions in Sweden and reasons behind possibilities to scale up to the entire Sweden user base of local news production line.

In a longer run the focus is also to use the categories metadata point in other products such as improving article personalization algorithms, data analytic of supply and demand of article categories. This article is an extension from Linden et al presented at Fedcsis conference 2018 [1]. The auto-categorization prediction accuracy are improved by parameter optimization since the last article and are now at 80 % and that is the underlying reason why the system is ready for an production environment. The classification model will categorize articles into different news topics and reply with a given confidence for each category. The while the journalists writes their articles in they would like to have categories in or close to real time.

2 Related Work

The novelty of this research, that concerns productification, is the idea to take the auto-categorization neural network

[*]Johannes Lindén, Holmgatan 10 Sundsall Sweden, 010-142 80 69 & johannes.linden@miun.se

model from its experimental phases all the way to a use case where it brings value. Several research projects has been done for managing and supporting big data pipelines, where live data needs to be processed by several organizations (for example analysis and business intelligence) [2]-[3]. The auto-categorization model perdictions needs a similar automatic data pipeline to uphold the load of each prediction and at the same time allow analysis of the result that will be brought back to the model in a feedback loop. Blaiszik et al made a scalable service that manages machine learning models through a structured API [4]. Blaiszik is using docker containers to wrap the model logic into an environment with controlled variables. This research approach is similar to Blaiszik but instead of using containers for only the models the idea is to have one container for the service as well. Cloud based services such as AWS Sagemaker or IBM's Watson has benefits as well as flaws explained in detail by Hummer et al that builds their own cloud service for training machine learning models [5]. According to Hummer's research it is essential for the AI expert to focus on the model itself and care less about the pipeline and how to set it up properly but Hummer also mentions the importance of having the settings available to tweak the service for special needs. For simplicity this research are using Sagemaker to deploy models in production since there is knowledge about other AWS services available and having all systems in the same cloud has other benefits of latency and authorization flexibility. An evaluation investigation in signal processing was issued that was written by Massimo Ravasi and Marco Mattavelli where they motivated the importance of the system complexity and the partitioning of hardware and software evaluation [6].

As for the auto-categorization model itself Payne et al have made an evaluation of their SVM classification algorithm which yields promising results in performance [7]. The model used in this research utilizes a bigger neural network to comprehend a larger problem domain. Rather than only focusing on a domain concerning digital archives it is focused on local news which could be very specific to each region of the country but could also be very general where regions have common information needs of local news.

3 Approach and Model

The proposed four step model in previous research that predicts categories of arbitrary text paragraphs are going to be the main model in this research that will conduct scalability experiments and propose a system structure of this machine learning model. See Figure 1 for an overview of previous implementation.

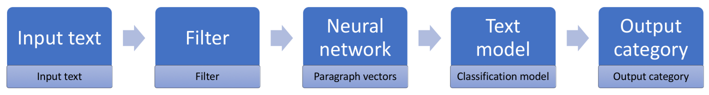

Figure 1: An overview of the model

The input is the algorithm parameters θ and a single document D, which is interpreted as a sequence of words $w_1, w_2, \cdots, w_n$. The output of the model is a set of category probabilities $c_1, c_2, \cdots, c_i, \cdots, c_m$ computed by 1.

$$c_i = P(\text{ith category}|D, \theta) \approx P(\text{ith category}|D) \tag{1}$$

In this research the model are used for example as shown in Figure 2, the article shown to the left in a web-browser will request a category from the trained auto-categorization model and retrieve a list of probable categories.

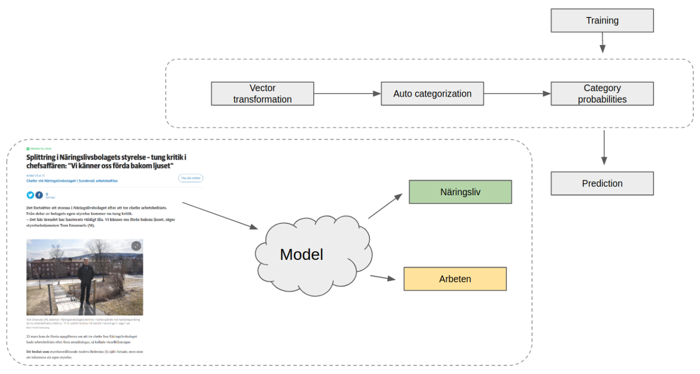
Training
Vector transformation
Auto categorization
Category probabilities
Prediction
Splittring i Näringslivsbolagets styrelse – tung kritik i chefsaffären: "Vi känner oss förda bakom ljuset"
Model
Näringsliv
Arbeten

Figure 2: Auto-categorization in a real case scenario. While a user navigates to a article page the category of that particular article is determined based on the body/text within it.

The algorithm takes an unstructured sequence of words forming a text paragraph as input. Language of this model is Swedish and even though the text length could be of any length the articles that has been evaluated has the length between 5 to 600 words in the first step of Figure 1. Text paragraphs are filtered before training to not include articles with only links or empty texts since these articles are not describing what the article is about. Within an text paragraph some additional modifications is done which is step two in Figure 1. Non-alphabetic characters, exclamation and question marks are replaced by full stops. XML tags and its attributes are removed. Some additional optional modifications are left in the settings file to be turned on or off while using the system. Such modifications include Named Entity Recognition (NER) substitution of locations, organizations, names and time units, another modification are Part-of-speech (POS) tagger and Dependency parser which will filter out certain types of tagged words and leaf nodes [8]. POS and Dependency parser chosen in this research where Google SyntaxNet [9, 10]. Out of the resources mentioned in Nilson et al, we selected a treebank made by Jan Einarsson's project, which is well documented [11]-[12]. The other steps in Figure 1 are described in the following sections.

3.1 Paragraph Vectors

The third step in Figure 1 is a constructed one layered neural network model that will transform the filtered text paragraphs from previous step into vectors. The paragraph vectors are unique vectors that mapps the paragraph to a high dimensional vector space. A softmax activation function makes sure that the elements are within the interval -1 to 1. The vectors are constructed in such a way that it is possible to relate a paragraph from the distance between another paragraph. The relation of two paragraphs can be obtained by computing the cosine similarity which will give a positive value when the documents are sharing similar contexts, a value close to zero when no relatedness could be found, and a negative number when the paragraphs appears in in opposite contexts [13]. The vectors can also use common subtraction and additions operations in case other related context will be retrieved, as shown in 2.

$$king - man + woman = queen \tag{2}$$

The paragraph vectors context awareness contains information about what makes a document category as shown in previous research [1]. The paragraph vectors are as before computed using the PV-DM algorithm which is an extension of the known word2vec algorithm bag of words (WV-BOW) [1, 14].

Experiments previously conducted showed that the PV-DBOW paragraph algorithm is reliable when implemented by the distributed memory vector concatenated with the distributed bag of words vector described by Mikolov et al [13].

3.2 Text Model

The paragraph representation vectors is input to the fourth step where the auto-categorization takes place. The categorization algorithms we proposed in previous research experiments where decision trees, random forest, multi layer perception and long-short term memory (LSTM). The one outperformed the others where the LSTM model which are used in this extended paper with the possibility to exchange to another model at any time through a settings parameter in the system. The output is the category belonging probabilities for each considered category described in 1. The LSTM model has a time parameter that each time sequence are used for a vector transformed sentence in the paragraph. Figure 3 shows the LSTM input format, each colored line is a sentence which are fed in per time-slot in the model for training .

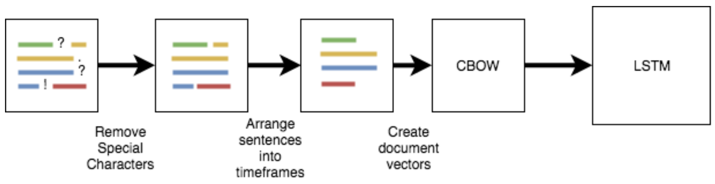

Figure 3: The input vector of the LSTM algorithm is shown above. It first applies the filter layer conditions, then divides the document into sentences to be used as input data for the CBOW algorithm that produces the document vectors.

LSTM models are based on the principles of recurrent neural networks (RNN). A RNN is constructed like a neural network with an input, hidden layers and a output layer. The RNN-cell can be visualised as shown in Figure 4. The activation function of an RNN is usually the *tanh* function. For each iteration, the model is trained by backpropagation through the network. The purpose of RNN is to have a short-term memory that remembers previous neurons. One of the first and simple constructions of RNN is the recurrent neural network language Model (RNN-LM). The hidden layer of an RNN-LM algorithm remembers the neurons one time-step back in the training history [15].

For different use-cases there are different variations of RNNs. Andrej Karpathy summarises the different networks that are used into different mappings [16]. One-to-one mapping is the original algorithm, for example the RNN-LM algorithm. One-to-many mapping when there is one input and several RNNs connecting to several outputs. This mapping can, for example be, used for image prediction with one image and several words that predict the image. Many-to-one mapping is where there are several inputs mapping to one output, this mapping can for example be used for classification. Many-to-many mapping is what Karpathy describes as two different mappings: one mapping that maps to an equal number of input and output RNNs (N-N), and another mapping that maps to a different number of inputs and outputs (N-M). The N-N mapping can, for example, be used to predict video sequences over time, while N-M mapping can be used for translation problems.

LSTM networks are a special case of RNN that tends to solve a problem in the original RNN. The long-term memory in RNN (the gradient descent) exponentially diverges to infinity or converges to zero in many cases and LSTM introduces an additional memory cell and forget parts of the information and therefore avoiding more cases of vanishing gradients. LSTM networks introduce three sigmoid layers and certain gates that only let parts of the information through to compensate for the vanishing gradient. The first sigmoid layer determines what information that is important from the previous LSTM-cell, the second sigmoid layer determines what information is important from the *tanh* layer in the current cell and the third sigmoid layer determines what information will be passed to the next LSTM-cell. The gates that open or close based on the input from the previous LSTM-cell either remove or add information to a cell state that is also passed through to the next LSTM-cell. The third sigmoid layer extracts a piece of information from the cell state to the output value [17, 18].

3.3 Output Category

Output category is the last, fifth, step of the model which determines the output categories. This step interprets the output of the categorization model and limits the categories to suggest via a threshold of the category probability. If the probability is higher than the average probability of seen paragraphs the category are recommended and sent to output of the model. Before the categories that will be replied is returned from the auto-categorization model the probabilities are normalized according to 3.

$$\frac{value - min(value)}{max(value) - min(value)} \tag{3}$$

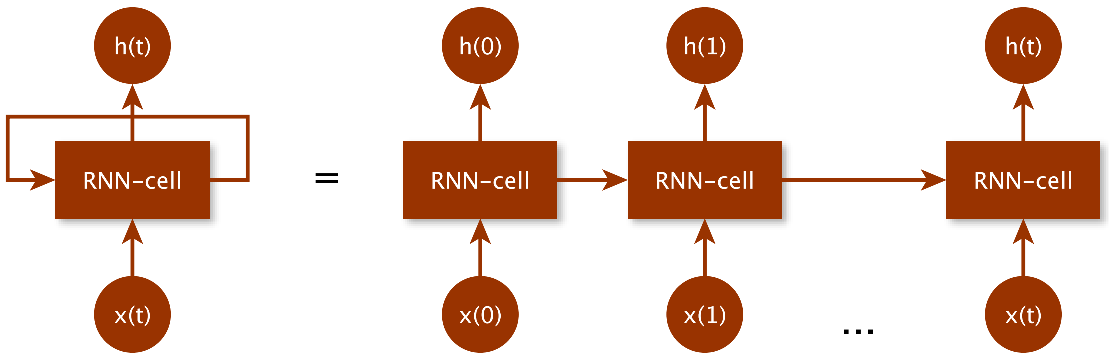

Figure 4: A recurrent neural network cell. The cell to the left is the general notation of the hidden layer in an RNN model. The right unfolded version is the representative of the RNN with a short-term length of t.

4 Architecture

The architectural approach consists of the system, the data pipeline and the neural network model. For making the auto-categorization available to use on texts written by the journalists all three pieces has to be in place. The procedure should be close to real time and this was accomplished by a system that could easily be scaled up to bigger machines with load distributed support for the user prediction requests. Figure 5 shows an overview of the system.

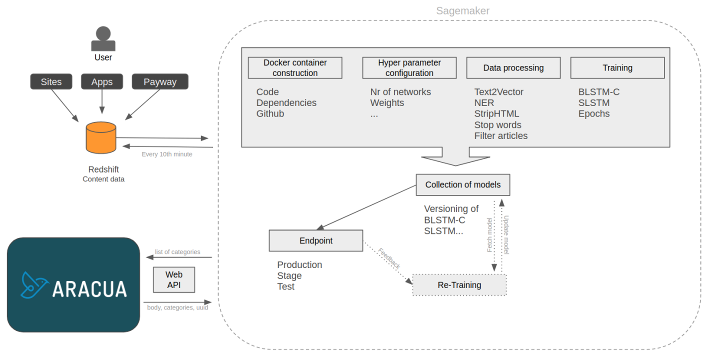

Figure 5: Architectural overview of production environment.

Aracua is the name of the system the journalists are writing their articles in, this system will request categories given the text of the article. Each categorization request is stored together with the previous category for further live evaluation purposes. The requests are stored in a AWS Redshift cluster to compare the final categories of each article with the suggested ones along the way while the journalists are writing the article [19]. In Redshift other services will have access to the auto-categorized categories to be used for example in personalization purposes. This section will further describe the deployment process of the model.

4.1 System

A Ngnix server was used for the system that can handle requests and reply the the model prediction of categories. The server instantiates python flask applications on demand when the load is high and one instance cannot keep up the clients requests as illustrated in Figure 6.

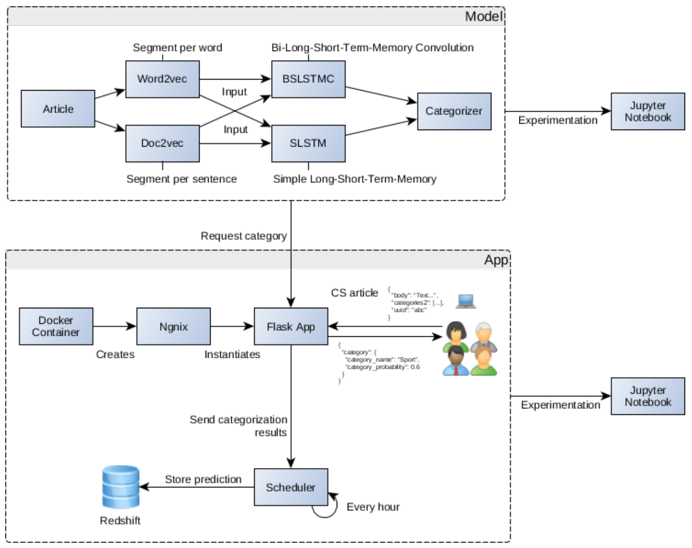

Figure 6: Architecture of the server system.

All runs inside a docker container to make it easy to increase the memory and processing power on an AWS EC2 instance if needed [20]. Each time a client requests categories for an article the user also sends the current categories on the article so that in a later step it is possible to evaluate an article over time as it is being worked on.

The predictions and articles are temporarily stored in memory to speed up the replies for the same article. A scheduled task is triggered every 10 minute or when number of articles in memory are larger than 1000 documents. The scheduler dumps these articles in memory to a persistent database where evaluation will be performed. The model is parameterized so that depending on the settings different models are being used for the prediction see Section 4.3.

4.2 Data pipeline

The system gets article contents that are being written by journalists from Aracua and stores them in a database. The articles are distributed on various of different places depending on the product that will use them. The auto-categorization product retrieves all articles and trains its models on these articles as described in Section 3. Each model that is trained is stored separately so that it is easy to switch models in production. A retraining process is started in production to dynamically improve the model for the articles. There are not only models for each environment (developement, staging and production) but also models for different versions and improved models as the time progresses.

4.3 Neural network model

The parameters that is used to train the model are stored in a configuration file. The configuration file contains what model structures that are going to be used, how long the training will take in terms of number of epochs, layer sizes pre-processing steps to be done, where the data is taken from and model stored to. Currently the following setup shown in Table 1 is used since it was shown to be the best one in previous experiments.

Table 1: Neural network model parameters

Param	Value
Layer size	3
LSTM timestep size	200
Document vector size	1000
Pre-processing steps	NER, StripHTML, Filter stop words, Part-of-speech

4.4 Hardware specifics

The evaluation conducted in this research is performed on an AWS Graviton Processor with two 64-bit Arm cores. The processor is caped at 2.3 GHz clock frequency, the memory is 16 GB and high network speed according to AWS specifications [21]. The computer used corresponds to the specification of ”ml.m4.xlarge” in Amazon Instance specifications[21].

5 Results

Figure 5 and 6 shows the resulting production system and how it works. The main result is that this system is used by thousands of journalists all over Sweden. Additional results of this extended article is to provide evaluation metrics of the system in terms of responsiveness to the journalists. Figure 7 shows how long the response time is in average given the number of clients trying to access the system at the same time.

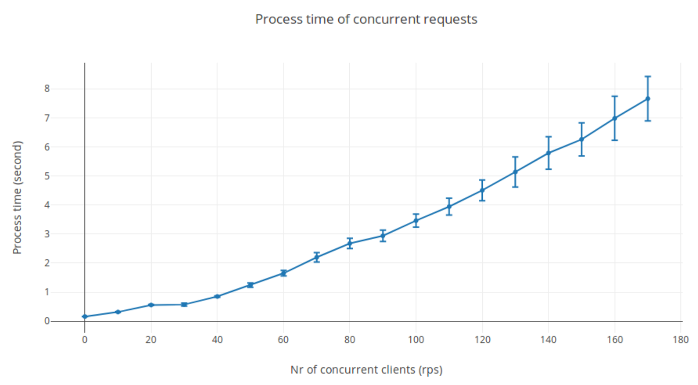

Figure 7: Process time of concurrent requests with standard deviation.

By analysing the number of characters an article body consists of when categorizing it is possible to say how much implications long articles have on the system, the resulting analysis can be seen in Figure 8.

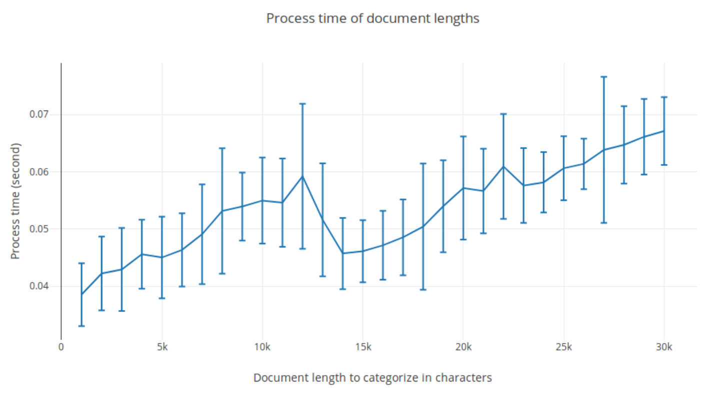

Figure 8: Process time of document lengths with standard deviation.

The training process is the process that initially takes the longest time of all in this research. Training a model with all articles of the categories in question takes up to 2.2 hours using a document vector embedding size of 600 elements as shown in Figure 9.

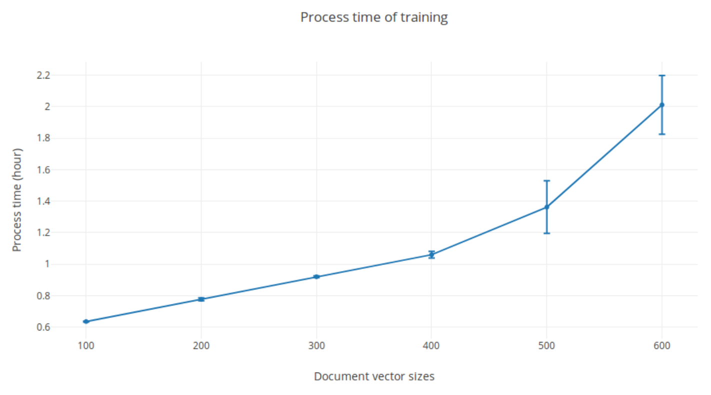

Figure 9: Process time of training with standard deviation.

6 Conclusion

In this article a proposition for a productification of a categorization algorithm that through a web API serves journalists with automatically categorized texts in their writing tool. The main question was if the performance of such model could be serving a large amount of users in their daily work and in that case how many users can it handle. It can be concluded that the number of concurrent requests have somewhat a linear time complexity but could potentially be exponential of higher orders for larger number of clients. Analysing the article length it can be interpreted to have an impact of the process time but would not be a problem for journalists since it is rare that they write long articles. Around 15 000 characters seem to be a good number of characters to have in an article. Training performance is growing with polynomial complexity and could be a risk as more categories are introduced and larger vector embedding needs to be considered. Future work for this project will be to include more categories with more complex algorithms which will use a wisdom of the many approach.

References

[1] Johannes Lindén, Stefan Forsström, and Tingting Zhang. Evaluating combinations of classification algorithms and paragraph vectors for news article classification, 2018.

[2] Ryan Chard, Kyle Chard, Jason Alt, Dilworth Y Parkinson, Steve Tuecke, and Ian Foster. Ripple: Home automation for research data management. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 389–394. IEEE, 2017.

[3] Jack Deslippe, Abdelilah Essiari, Simon J Patton, Taghrid Samak, Craig E Tull, Alexander Hexemer, Dinesh Kumar, Dilworth Parkinson, and Polite Stewart. Workflow management for real-time analysis of lightsource experiments. In *2014 9th Workshop on Workflows in Support of Large-Scale Science*, pages 31–40. IEEE, 2014.

[4] Ben Blaiszik, Kyle Chard, Ryan Chard, Ian Foster, and Logan Ward. Data automation at light sources. In *AIP Conference Proceedings*, volume 2054, page 020003. AIP Publishing, 2019.

[5] Waldemar Hummer, Vinod Muthusamy, Thomas Rausch, Parijat Dube, and Kaoutar El Maghraoui. Modelops: Cloud-based lifecycle management for reliable and trusted ai.

[6] Massimo Ravasi and Marco Mattavelli. High-level algorithmic complexity evaluation for system design. *Journal of Systems Architecture*, 48(13-15):403–427, 2003.

[7] Nathaniel Payne and Jason R Baron. Auto-categorization methods for digital archives. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 2288–2298. IEEE, 2017.

[8] Atro Voutilainen. Part-of-speech tagging. *The Oxford handbook of computational linguistics*, pages 219–232, 2003.

[9] Daniel Andor, Chris Alberti, David Weiss, Aliaksei Severyn, Alessandro Presta, Kuzman Ganchev, Slav Petrov, and Michael Collins. Globally normalized transition-based neural networks. *arXiv preprint arXiv:1603.06042*, 2016.

[10] Chris Alberti, Daniel Andor, Ivan Bogatyy, Michael Collins, Dan Gillick, Lingpeng Kong, Terry Koo, Ji Ma, Mark Omernick, Slav Petrov, et al. Syntaxnet models for the conll 2017 shared task. *arXiv preprint arXiv:1703.04929*, 2017.

[11] Jens Nilsson and Johan Hall. *Reconstruction of the Swedish Treebank Talbanken.* Matematiska och systemtekniska institutionen, 2005.

[12] Jan Einarsson. Talbankens talsprkskonkordans. 1976.

[13] Quoc V Le and Tomas Mikolov. Distributed Representations of Sentences and Documents. In *ICML*, volume 14, pages 1188–1196, 2014.

[14] Johannes Lindén. Understand and Utilise Unformatted Text Documents by Natural Language Processing algorithm. 46(0), 2017.

[15] Tomáš Mikolov, Stefan Kombrink, Lukáš Burget, Jan Černocky, and Sanjeev Khudanpur. Extensions of recurrent neural network language model. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5528–5531. IEEE, 2011.

[16] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. *Andrej Karpathy blog*, 2015.

[17] Christopher Olah. Understanding lstm networks. *GITHUB blog, posted on August*, 27:2015, 2015.

[18] Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. LSTM Neural Networks for Language Modeling. In *Interspeech*, pages 194–197, 2012.

[19] Joseph Baron and Sanjay Kotecha. Storage options in the aws cloud. *Amazon Web Services, Washington DC, Tech. Rep*, 2013.

[20] Gideon Juve, Ewa Deelman, Karan Vahi, Gaurang Mehta, Bruce Berriman, Benjamin P Berman, and Phil Maechling. Scientific workflow applications on amazon ec2. In *2009 5th IEEE international conference on e-science workshops*, pages 59–66. IEEE, 2009.

[21] Amazon Co. Amazon SageMaker Instance Types - Amazon Web Services (AWS). https://aws.amazon.com/sagemaker/pricing/instance-types/.