TITLE

Bachelorarbeit

der Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

vorgelegt von

Sven Kellenberger

01.08.2018

Leiter der Arbeit:
TITLE NAME
Institut für Informatik

Abstract

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer

Contents

Chapter 1

Introduction

1.1 Motivation

Automatic text correction is a ubiquitous technology in our today world. Every smartphone, every word processing software, every browser provides some form of spelling error detection and correction for text input. These systems usually rely on a dictionary of correct words [6, 5] or some machine learning algorithm [20] to find errors and possible corrections.

Source code is very sensitive to errors of all kinds and therefore this functionality is also desirable for code editors. The syntax of a programming language is strictly defined which enables integrated development environments (IDEs) to detect syntax errors before the program is even run. Of course the possibilities of an IDE are limited by the properties of the programming language, e.g. is it strongly typed or weakly typed. However, the error detection in source code is mostly limited to syntax errors, while semantic and logical errors show only at runtime or sometimes go completely unnoticed. These kinds of errors are also the hardest ones to fix. In a strongly typed language like Java, a lot of possible errors in naming and accessing attributes can be eliminated, because each variable has to be initiated before it is used and the type of the variables is known at all times and therefore also their available attributes and methods. In weakly typed languages like Ruby however, one can not determine what type of object a variable holds before runtime. This creates additional sources of runtime errors.

While syntax errors can be detected by a suitable algorithm, traditional algorithms can only hope to help prevent semantic and logical errors. This is where machine learning algorithms could step in. In the past years, deep neural networks have proven to be very effective in learning generalised concepts and applying them to single cases. For example in [17] a network is trained to transfer the style of a painting to a video sequence. That's why it should also be possible to train a network to recognize and correct certain logic errors in source code.

The aim of this project is it to train a character based sequence-to-sequence model on the task of source code correction. The implementation of the model is based on the neural machine translation (NMT) model provided by Tensorflow [12]. As a dataset the Java Github Corpus [1] is used as a source of correct

data. This data is then perturbed as random syntax, semantic and logic errors are added. The performance of different model architectures is then evaluated for the introduced errors.

1.2 Outline

This thesis is divided into five chapters, including this introduction. In the second chapter an overview of the prior work on the task of spelling checking and correction is given and also some work on error detection and correction in source code. Furthermore a short introduction to the machine learning techniques and architectures used in this thesis is given. The third chapter provides a description of the used model, the training procedure and the dataset construction. In chapter four the experiments are explained and the optained results analysed. Chapter five provides the conclusion and suggestions for future work.

Chapter 2

Background

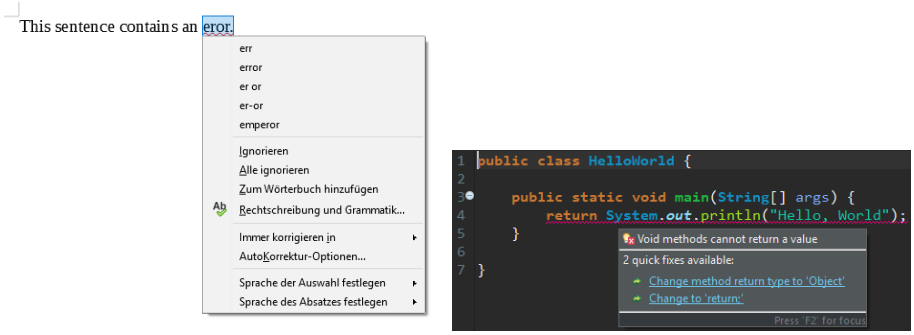
2.1 Origins of Spelling Checkers and Correctors

With the emergence of word processing programs and the following digitalization of text documents, spelling checkers and correctors have become a common helper in our everyday lives. However, the research on this topic has begun much earlier [15].

The original motivation for a spelling checker was to find input errors in databases. For example, in [3] the authors aim to find incorrectly spelled names, dates and places in a genealogical database. This is done by computing the frequency of trigrams (three sequential characters) in the source text and based on that the probability of a character given some context, i.e. its adjacent characters. Erroneous words are found by looking at its trigrams. If a word consist of a number of unusual character combinations, it is probably spelled wrongly. However, it is easy to see that this method is not very useful for new, rare or foreign expressions like "doppelgänger" and for typos with high probabilities of being correct. Furthermore the model is limited to the vocabulary used in the text.

These problems were solved by the introduction of dictionaries. A dictionary is a list of correctly spelled words which can optionally be extended by the user. For every word, the program checks if it is part of the dictionary. If it is, then it is spelled correctly, otherwise there is an error. This method was enhanced by the addition of direct user interaction. Instead of outputing a list of incorrect words, the program would show the user the words it assumed to be incorrectly spelled and then give the user some possible actions to chose from. Of course this method is still not perfect, for example if "know" is misspelled as "now", no error is indicated even though it could be concluded from the context that a verb is expected in this place.

Spelling correction is another enhancement of these methods. In [6] a dictionary is used to find incorrect words. It is assumed that these words contain only one of four types of errors: one character was wrong, one extra character was inserted, one character was missing or two adjacent characters were transposed. Under this assumption the dictionary is searched for possible corrections. In [5] the author uses a dictionary to find incorrect words as well. For the found words he uses digrams (similar to the trigrams mentioned above, just with two



(a) A subfigure (b) A subfigure

Figure 2.1: A figure with two subfigures

instead of three characters) to suggest corrections for incorrect words.

2.1.1 Modern Approaches

2.2 Automatic Correction of Source Code

Chapter 3

Model and Training

3.1 Components

3.1.1 LSTM

A recurrent neural network (RNN)[19] is a special form of neural network that is used for sequential tasks. It works by having multiple copies of the network, one for each timestep. As the input proceeds in time, each network passes information to it's next instance as seen in INSERT FIGURE HERE. For an input sequence $(\mathbf{x}_1, ..., \mathbf{x}_n)$ the RNN produces at each timestep t a hidden state vector $\mathbf{h}_t$ as follows:

$$\mathbf{h}_t = \tanh\left(\mathbf{W}\begin{pmatrix}\mathbf{x}_t \\ \mathbf{h}_{t-1}\end{pmatrix}\right)$$

However, RNNs have proven to be hard to train, especially on long-range dependencies [8]. In theory, they should be able to deal with these dependencies but either vanishing or exploding gradients usually prevent them from doing so. To solve this issue, Long Short-Term Memory networks (LSTMs) [9] were proposed. In addition to $\mathbf{h}_t$, LSTMs also pass a memory state vector $\mathbf{c}_t$ to the next instance as can be seen in INSERT FIGURE HERE. The LSTM can choose at each timestep if it wants to read or forget information from the memory vector or write new information onto the vector. This is done by using explicit gating mechanisms:

$$\mathbf{f}_t = \sigma\left(\mathbf{W}_f\begin{pmatrix}\mathbf{x}_t \\ \mathbf{h}_{t-1}\end{pmatrix}\right) \qquad \mathbf{i}_t = \sigma\left(\mathbf{W}_i\begin{pmatrix}\mathbf{x}_t \\ \mathbf{h}_{t-1}\end{pmatrix}\right)$$
$$\mathbf{o}_t = \sigma\left(\mathbf{W}_o\begin{pmatrix}\mathbf{x}_t \\ \mathbf{h}_{t-1}\end{pmatrix}\right) \qquad \mathbf{g}_t = \tanh\left(\mathbf{W}_g\begin{pmatrix}\mathbf{x}_t \\ \mathbf{h}_{t-1}\end{pmatrix}\right)$$

where σ is the sigmoid function. $\mathbf{f}_t$, $\mathbf{i}_t$ and $\mathbf{o}_t$ can be thought of as binary gates that decide which information from $\mathbf{c}_{t-1}$ should be deleted, which information of $\mathbf{c}_{t-1}$ should be updated and which information from $\mathbf{c}_t$ should be written to $\mathbf{h}_t$. Finally $\mathbf{g}_t$ is a vector of possible values that (gated by $\mathbf{i}_t$) can be added to $\mathbf{c}_{t-1}$ and because of the tanh in the equation its values may range from `-1` to `1`. The state vectors are then updated as follows:

$$\mathbf{c}_t = \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t$$
$$\mathbf{h}_t = \mathbf{o}_t \odot \tanh(\mathbf{c}_t)$$

Almost all remarkable results that are achieved today are achieved using either LSTMs or networks with a similar architecture like Gated Recurrent Units (GRUs) [4] because they are easier to train and excel at capturing long range dependencies.

3.1.2 The Sequence-to-Sequence Model

Traditional Deep Neural Networks (DNNs) process the whole input and then calculate some output, e.g. process an image and then classify it. This works well for problems where the input and the output are of a fixed dimension, however it is not suitable for problems where the input and the output are sequences of variable length. An example would be the input of a question and the network should produce an answer. We have seen that we can use LSTMs to process input sequences of variable length. However, in this case we want to process the whole input sequence and all the information that comes with it and only then start generating an output sequence. These problems are called sequence to sequence problems.

In [18] the Sequence-to-Sequence Model is introduced as a solution to these problems. The model was applied to the task of Neural Machine Translation (NMT) and has since become the state of the art architecture in this field. The main concept can be seen in FIGURE X. First the whole input sequence is fed into the network and the output is ignored. Then we input an end-of-sequence token `<EOS>` which signals the network to start producing the output. From there on the produced output tokens are fed to the network until the an end-of-sequence token is generated, thus signaling the end of the sequence. To speed up training the expected output is fed back to the network and not the actual produced output.

This architecture is further improved by splitting the network into two separate LSTMs (FIGURE). The first network takes all the input and encodes it into a vector which is then used to initialize the second network. It is first fed a start token `<GO>` and then the generated output until the end of the sequence is reached.

3.1.3 Attention-Mechanism

Attention is a relatively new concept for neural networks. The idea is to allow the network to chose on which information to focus at any given moment. For example in [14] attention is used on the task of high resolution image classification. These kind of networks often struggle with memory constraints and attention can help them to only load the significant part of the image into the memory.

Attention has subsequently been applied to NMT [13, 2]. The vector into which the input is encoded in the Sequence-to-Sequence model has been identified as a bottleneck which cuts down performance because of its limited capacity. After all the vector is of fixed dimensionality and needs to encode information

about the whole input sequence. Because of that attention is used as a mean for the decoder to peek at previous hidden states of the encoder. This is done via a context vector $\tilde{\mathbf{c}}_t$ which is combined with the current hidden state of the decoder $\mathbf{h}_t$. The resulting attentional hidden state $\tilde{\mathbf{h}}_t$ is then used by the decoder to generate the next output.

$$\tilde{\mathbf{h}}_t = \tanh\left(\mathbf{W}_c \begin{pmatrix} \tilde{\mathbf{c}}_t \\ \mathbf{h}_t \end{pmatrix}\right)$$

For the derivation of the context vector $\tilde{\mathbf{c}}_t$ all hidden states of the encoder $\bar{\mathbf{h}}_s$ are considered. For this an alignment vector $\mathbf{a}_t$, whose size equals the input sequence length, is calculated from the current decoder hidden state $\mathbf{h}_t$ and the encoder hidden states $\bar{\mathbf{h}}_s$. The values of a_t are then normalized using the `softmax` function.

$$a_t(s) = \frac{\exp(\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_s))}{\sum_{s'} \exp(\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_{s'}))}$$

Here, score is a content-based function used to compare the decoder hidden state $\mathbf{h}_t$ with each of the encoder hidden states $\bar{\mathbf{h}}_s$. There are various possible choices for this function, for example:

$$\text{score}(\mathbf{h}_t, \bar{\mathbf{h}}_s) = \begin{cases} \mathbf{h}_t^\intercal \mathbf{W}_a \bar{\mathbf{h}}_s \\ \mathbf{v}_a^\intercal \tanh\left(\mathbf{W}_a \begin{pmatrix} \mathbf{h}_t \\ \bar{\mathbf{h}}_s \end{pmatrix}\right) \end{cases}$$

The context vector $\mathbf{c}_t$ is then calculated as the weighted average over the encoder hidden states.

$$\mathbf{c}_t = \sum_{s'} a_t(s') \bar{\mathbf{h}}_{s'}$$

3.2 Implementation

3.3 Dataset Construction

Chapter 4

Experiments and Results

Chapter 5

Conclusion

5.1 Future Work

Appendix A

Appendix

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi. Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat.

Ut wisi enim ad minim veniam, quis nostrud exerci tation ullamcorper suscipit lobortis nisl ut aliquip ex ea commodo consequat. Duis autem vel eum iriure dolor in hendrerit in vulputate velit esse molestie consequat, vel illum dolore eu feugiat nulla facilisis at vero eros et accumsan et iusto odio dignissim qui blandit praesent luptatum zzril delenit augue duis dolore te feugait nulla facilisi.

Nam liber tempor cum soluta nobis eleifend option congue nihil imperdiet doming id quod mazim placerat facer

List of Tables

List of Figures

Bibliography

[1] Allamanis, Miltiadis ; Sutton, Charles: Mining Source Code Repositories at Massive Scale using Language Modeling. In: *The 10th Working Conference on Mining Software Repositories* IEEE, 2013, S. 207–216

[2] Bahdanau, Dzmitry ; Cho, Kyunghyun ; Bengio, Yoshua: Neural Machine Translation by Jointly Learning to Align and Translate. In: *CoRR* abs/1409.0473 (2014). http://arxiv.org/abs/1409.0473

[3] Carlson, Gary: Techniques for Replacing Characters That Are Garbled on Input. In: *Proceedings of the April 26-28, 1966, Spring Joint Computer Conference*. New York, NY, USA : ACM, 1966 (AFIPS '66 (Spring)), 189–193

[4] Cho, Kyunghyun ; Merrienboer, Bart van ; Gülçehre, Çaglar ; Bougares, Fethi ; Schwenk, Holger ; Bengio, Yoshua: Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In: *CoRR* abs/1406.1078 (2014). http://arxiv.org/abs/1406.1078

[5] Cornew, Ronald W.: A statistical method of spelling correction. In: *Information and Control* 12 (1968), Nr. 2, 79 - 93. http://dx.doi.org/https://doi.org/10.1016/S0019-9958(68)90201-5. – DOI https://doi.org/10.1016/S0019–9958(68)90201–5. – ISSN 0019–9958

[6] Damerau, Fred J.: A Technique for Computer Detection and Correction of Spelling Errors. In: *Commun. ACM* 7 (1964), März, Nr. 3, 171–176. http://dx.doi.org/10.1145/363958.363994. – DOI 10.1145/363958.363994. – ISSN 0001–0782

[7] Ghosh, Shaona ; Kristensson, Per O.: Neural Networks for Text Correction and Completion in Keyboard Decoding. In: *CoRR* abs/1709.06429 (2017). http://arxiv.org/abs/1709.06429

[8] Hochreiter, Sepp: Untersuchungen zu dynamischen neuronalen Netzen. (1991), 04

[9] Hochreiter, Sepp ; Schmidhuber, Jürgen: Long Short-Term Memory. In: *Neural Comput.* 9 (1997), November, Nr. 8, 1735–1780. http://dx.doi.org/10.1162/neco.1997.9.8.1735. – DOI 10.1162/neco.1997.9.8.1735. – ISSN 0899–7667

[10] Karpathy, Andrej ; Johnson, Justin ; Li, Fei-Fei: Visualizing and Understanding Recurrent Networks. In: *CoRR* abs/1506.02078 (2015). http://arxiv.org/abs/1506.02078

[11] Lévy, J. P.: Automatic correction of syntax-errors in programming languages. In: *Acta Informatica* 4 (1975), Sep, Nr. 3, 271–292. http://dx.doi.org/10.1007/BF00288730. – DOI 10.1007/BF00288730. – ISSN 1432–0525

[12] Luong, Minh-Thang ; Brevdo, Eugene ; Zhao, Rui: Neural Machine Translation (seq2seq) Tutorial. In: *https://github.com/tensorflow/nmt* (2017)

[13] Luong, Minh-Thang ; Pham, Hieu ; Manning, Christopher D.: Effective Approaches to Attention-based Neural Machine Translation. In: *CoRR* abs/1508.04025 (2015). http://arxiv.org/abs/1508.04025

[14] Mnih, Volodymyr ; Heess, Nicolas ; Graves, Alex ; Kavukcuoglu, Koray: Recurrent Models of Visual Attention. In: *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*. Cambridge, MA, USA : MIT Press, 2014 (NIPS'14), 2204–2212

[15] Peterson, James L.: Computer Programs for Detecting and Correcting Spelling Errors. In: *Commun. ACM* 23 (1980), Dezember, Nr. 12, 676–687. http://dx.doi.org/10.1145/359038.359041. – DOI 10.1145/359038.359041. – ISSN 0001–0782

[16] Reps, Thomas ; Teitelbaum, Tim ; Demers, Alan: Incremental Context-Dependent Analysis for Language-Based Editors. In: *ACM Trans. Program. Lang. Syst.* 5 (1983), Juli, Nr. 3, 449–477. http://dx.doi.org/10.1145/2166.357218. – DOI 10.1145/2166.357218. – ISSN 0164–0925

[17] Ruder, Manuel ; Dosovitskiy, Alexey ; Brox, Thomas: Artistic style transfer for videos and spherical images. In: *CoRR* abs/1708.04538 (2017). http://arxiv.org/abs/1708.04538

[18] Sutskever, Ilya ; Vinyals, Oriol ; Le, Quoc V.: Sequence to Sequence Learning with Neural Networks. In: *CoRR* abs/1409.3215 (2014). http://arxiv.org/abs/1409.3215

[19] Werbos, P. J.: Backpropagation through time: what it does and how to do it. In: *Proceedings of the IEEE* 78 (1990), Oct, Nr. 10, S. 1550–1560. http://dx.doi.org/10.1109/5.58337. – DOI 10.1109/5.58337. – ISSN 0018–9219

[20] Xie, Ziang ; Avati, Anand ; Arivazhagan, Naveen ; Jurafsky, Dan ; Ng, Andrew Y.: Neural Language Correction with Character-Based Attention. In: *CoRR* abs/1603.09727 (2016). http://arxiv.org/abs/1603.09727

Erklärung

gemäss Art. 28 Abs. 2 RSL 05

Name/Vorname: .....

Matrikelnummer: .....

Studiengang: ……

Bachelor ☐ Master ☐ Dissertation ☐

Titel der Arbeit: .....

.....

.....

LeiterIn der Arbeit: .....

.....

Ich erkläre hiermit, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen benutzt habe. Alle Stellen, die wörtlich oder sinngemäss aus Quellen entnommen wurden, habe ich als solche gekennzeichnet. Mir ist bekannt, dass andernfalls der Senat gemäss Artikel 36 Absatz 1 Buchstabe o des Gesetztes vom 5. September 1996 über die Universität zum Entzug des auf Grund dieser Arbeit verliehenen Titels berechtigt ist.

.....
Ort/Datum

.....
Unterschrift