Detecting Noisy Swiss German Web Text Using RNN- and Rule-Based Techniques

Janis Goldzycher*
Institute of Computational Linguistics
University of Zurich
janis.goldzycher@uzh.ch

Jonathan Schaber*
Institute of Computational Linguistics
University of Zurich
jonathan.schaber@uzh.ch

Abstract

This paper presents the system we submitted to the Swiss German language detection shared task, part of the GermEval 2020 Campaign, held at the SwissText & KONVENS 2020 conference. The goal of the task is to identify if a given text snippet is written in Swiss German. Our approach includes a reformulation of a binary to a multi-way classification problem, a character filter, a neural RNN-based classifier, and the addition of synthetic noise to the training set. The official evaluation of our submitted system results in an F1 score of 96.8%, achieving the second place in this shared task.

1 Introduction

In this paper we describe our approach and results for the Swiss German language detection shared task (GSWID 2020) at the SwissText & KONVENS conference. The objective of the shared task is to construct a system to automatically identify Swiss German (GSW) text snippets.

Generally, language identification has been viewed as a solved problem "suitable for undergraduate instruction", as McNamee (2005) deprecatingly remarks in the title of his paper. However, it is not clear if this view holds true for text snippets that are (1) short, (2) noisy, (3) from multiple domains, (4) written in a scarce resource language, or (5) which consist of non-standardized dialects (Gamallo et al., 2014; Jauhiainen et al., 2019). Since this shared task is about GSW language identification and uses tweets as test data, it combines all of these difficulties.

Previous approaches based on classical machine learning typically utilize character level features like single characters, character combinations (n-grams) and capitalization together with models such as naive bayes classifiers, support vector machines, and decision trees (Gamallo et al., 2014; Hanif et al., 2007; Kumar et al., 2015; Porta, 2014; Zubiaga et al., 2014). There have been both CNN-based (Jaech et al., 2016a,b; Li et al., 2018) and RNN-based (Jurgens et al., 2017; Kocmi and Bojar, 2017) neural approaches to language identification using character embeddings as representations, sometimes with additional features incorporated, like n-grams (Chang and Lin, 2014) or word embeddings (Samih et al., 2016). We approach this problem using a bidirectional GRU (BiGRU) architecture similar to the one put forward by Kocmi and Bojar (2017).

In this paper we describe our system, comprising: (1) a reformulation of a binary to a multi-way classification problem, (2) a BiGRU-based neural architecture, (3) a character-based filter, and (4) a noisifier module.

2 Data

Provided Data The shared task organizers provide a list of approximately 2,000 GSW tweets to be used as positive training examples.[1] The use of further training material is explicitly allowed and encouraged. In the following paragraphs we give a review of additionally collected data.

Swiss German We collect GSW data from the following sources: the NOAH corpus (Hollenstein

*Equal Contribution.

[1] Due to the distribution regulations of Twitter, the organizers published only tweet IDs. At the time of downloading, 22 of these tweets were not available anymore, so the actual number of tweets we are able to use is 1,978.

and Aepli, 2014), a collection of texts from various genres; the Swisscrawl corpus (Linder et al., 2019), which consists of user entries from forums and social media; the chatmania data from the SpinningBytes corpus (Grubenmann et al., 2018), containing forum entries; and the GSW corpus from the corpus collection of the University of Leipzig (Goldhahn et al., 2012), that also incorporates web data, mainly from chat forums.

Other Languages There is of course an abundant amount of textual data in a multitude of other languages which cannot be entirely considered, or feasibly be included in a training set. We devise the following difficulty-scale from A (easy) to D (difficult) as a prioritization guideline as for which languages we presume are hard to distinguish from GSW and thus most important to include in the training set as negative examples:

A: languages written in non-GSW character sets[2] (e.g. Chinese, Hindi, Arabic)
B: languages written in scripts that overlap with the GSW character set (e.g. Afrikaans, Tagalog, English, Tok Pisin)
C: languages in B that share parts of the lexicon with GSW (e.g. English, Italian, French, Standard German)
D: languages and varieties in C that are closely related to GSW (e.g. Standard German, Dutch, English, Bavarian)

Note that the following set memberships hold: $B \supset C \supset D$ and $A \cap B = \emptyset$.

We only collect languages from B, with special focus on C and D, since text snippets written in a language from A can be filtered out in a rule-based manner.

We collect data for all languages from the aforementioned corpus collection of the University of Leipzig. For Standard German, we additionally gather texts from the Hamburg Dependency TreeBank (Foth et al., 2014). For all corpora that are not comprised of tweet-like text, we treat each sentence as an individual text snippet. An overview of our collected data is shown in table 1. We split our data set with a ratio of 0.95/0.05 resulting in a training set containing 3,605,283 instances and a development set of 189,752 instances.

[2]We define the *GSW character set* as the set of characters found on a GSW keyboard. This differs slightly from e.g. a Standard German Keyboard, which lacks characters like "è", "à" and "é".

Language	# instances	relative
Swiss German (GSW)	780,502	19.17%
Standard German	568,493	13.97%
English	304,822	7.49%
Italian	300,077	7.37%
Dutch	300,043	7.37%
Swedish	300,002	7.37%
Luxembourgish	300,000	7.37%
Norwegian	300,000	7.37%
French	299,017	7.35%
Low German	100,000	2.46%
West Frisian	100,000	2.46%
Portuguese	100,000	2.46%
Romanian	100,000	2.46%
Tagalog	100,000	2.46%
Bavarian	30,000	0.74%
Lombard	30,000	0.74%
Yiddish	30,000	0.74%
Croatian	10,001	0.25%
Northern Frisian	10,000	0.25%
Other	8,060	0.20%
Total	4,071,017	100.00%

Table 1: Overview of collected text snippets per language. Languages with less than 1,000 examples, e.g. Turkish, are subsumed under class *other*.

Noise Through manual inspection of the tweets that were provided as training data we observed that they are significantly noisier than the rest of our training data.

We identify two kinds of noise in this data: token-level noise and character level noise. Both can be produced on purpose or by accident. Token-level noise consists of words, phrases or citations in other languages, mainly English or Standard German, in otherwise GSW tweets. Character-level noise consists of omissions, insertions or repetitions of single characters. Examples can be found in Appendix B.

3 Method

Task Formalization We formalize the task as follows: Assign a label $y \in \{0, 1\}$ to an input sequence of characters $x = \{x_0, x_1, x_2, ..., x_n\}$, where 0 corresponds to the class *swiss german* and 1 corresponds to the class *not-swiss german*.

However, the *not-swiss german* class is a very broad category since it not only contains all other languages, some of which are similar to GSW, but also all possible string sequences that do not appear in GSW. Thus, we hypothesize that more fine-grained labels will lead to more homogeneous and better separable classes.

Following this line of reasoning, we define three different granularity levels: binary, ternary and fine-grained. The binary setting corresponds to the task formalization described above. In the ternary

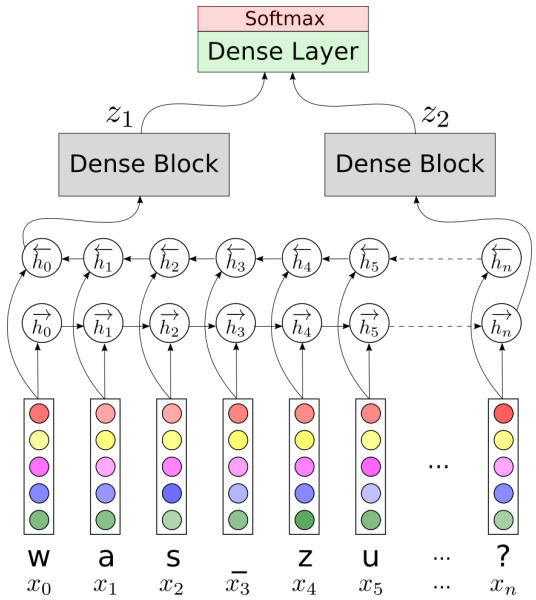

Figure 1: Neural Architecture based on character embeddings, a BiGRU, two dense blocks and a dense layer.

setting we split the class *not-swiss german* into the classes *standard german* and *other*. And in the fine-grained setting, each language present in table 1 corresponds to one class, with an additional class *other*. For our collected data set, this leads to a total of 23 classes.

Pipeline We construct a pipeline where an incoming text snippet is first cleaned of hashtags, mentions and URLs. Then a rule-based character filter decides if the text snippet is a member of A and if so, immediately classifies the text snippet as *not-swiss german*. If the text snippet is not part of A, it might be an instance of *swiss german* and hence is clipped to a prespecified length, which we treat as a hyper parameter, and fed into a neural classifier.

During training time, we make two modifications to the pipeline: (1) The rule-based character filter is left out because our data only consists of text snippets from languages in B. (2) We make use of an additional noisifier, which adds noise specifically modeled after the noise that is actually encountered in GSW text snippets on the web. In the rest of this section we describe the main parts of the pipeline in detail.

Character-Based Filter For a given sequence of characters x, the character-based filter computes the relative frequency of characters in x that do not appear in the GSW character set. If this frequency surpasses a given threshold, x is labeled as *not-swiss german*.

Neural Model Our neural model comprises character embeddings, a BiGRU (Cho et al., 2014), two blocks of dense layers and a final dense layer. The BiGRU takes the embedded characters as input and produces the outputs $\overrightarrow{o_0}, ..., \overrightarrow{o_n}$ and also a last hidden state $\overrightarrow{h_n}$ for the forward GRU. For the backward GRU we get $\overleftarrow{o_n}, ...\overleftarrow{o_0}$ and $\overleftarrow{h_0}$ respectively.

We ignore all BiGRU outputs and only use the last hidden states $\overrightarrow{h_n}$ and $\overleftarrow{h_0}$.[3]

Each hidden state is fed into a block of two dense layers with dropout before both layers and the rectified unit linear function in between. The outputs of the two dense blocks z_1 and z_2 are concatenated and fed into a final dense layer with the number of classes as the output dimension. We apply a log-softmax function to the output to turn the neural activations into a probability distribution over the target classes. Note that the number of target classes depends on the chosen level of granularity.

For optimization we use the negative log likelihood loss combined with the Adam optimizer (Kingma and Ba, 2014). We initialize the character embeddings randomly and train them jointly with the rest of the model.

Noisifier Based on the assumption that the test data has a similar amount of noise as the tweets provided for training, we introduce a noisifier with the goal of injecting this type of noise into the entire training data, which contains large amounts of text snippets from "clean" resources like news texts. We refer to this difference in noise between corpora as *noisiness gap*. Recall that we observed token-level and character-level noise in the training data in section 2. In what follows, we will address both types of noise separately.

For the token level noise we created a handcrafted list L consisting of English and Standard German words often found in GSW tweets, comments and messages. Additionally, we add mentions of Swiss locations to L.[4]

[3] In earlier experiments we also used the BiGRU outputs by concatenating them with the last hidden states and then fed this entire feature vector into dense layers. However, we found that using these outputs decreased performance.

[4] We try to avoid that the model learns to associate Swiss location names with GSW text which presumably would lead to false positives.

Configuration & Training					Development Set						Test Set			
Emb	Clip	G	N	TT	Prec	Rec	Acc	F1	AccT	AccF	Prec	Rec	Acc	F1
100	280	b	f	15.7	0.946	0.926	0.976	0.936	-	-	0.898	0.920	0.911	0.909
100	280	t	f	15.9	0.971	0.957	0.986	0.964	0.980	-	0.905	0.942	0.924	0.923
100	280	f	f	15.8	**0.994**	**0.992**	**0.997**	**0.993**	0.989	**0.994**	0.907	0.990	0.946	0.947
100	100	b	f	6.8	0.987	0.980	0.994	0.983	-	-	0.872	0.880	0.880	0.876
100	100	t	f	6.6	0.982	0.973	0.992	0.978	0.988	-	0.932	0.954	0.944	0.943
100	100	f	f	**6.5**	0.991	0.989	0.996	0.990	0.988	0.991	0.930	0.984	0.957	0.956
300	100	b	f	7.0	0.987	0.977	0.993	0.981	-	-	0.949	0.931	0.943	0.940
300	100	t	f	7.1	0.992	0.988	0.996	0.990	**0.995**	-	**0.959**	0.948	0.955	0.953
300	100	f	f	7.1	0.992	0.989	0.996	0.990	0.988	0.992	0.927	0.985	0.956	0.955
300	100	b	t	8.4	0.993	0.987	0.996	0.991	-	-	0.955	0.980	0.968	0.967
300	100	t	t	7.8	0.993	0.986	0.996	0.990	**0.995**	-	0.947	0.983	0.965	0.965
300	100	f	t	7.8	**0.994**	0.987	**0.997**	0.991	0.988	0.992	0.945	**0.993**	**0.969**	**0.968**

Table 2: Results on the development and test set. Abbrevations: **Emb**edding dimension, **Clip**ped after *m* characters, **G**ranularity (**b**inary, **t**ernary, **f**inegrained), **N**oise injected (**t**rue, **f**alse), **T**raining **T**ime in hours. The last row shows the configuration that we submitted to the shared task.

The token-level noisifier receives as input a clean training example x consisting of k tokens and the two thresholds $p_1 \in [0, 1]$ and $p_2 \in [0, 1)$ with $p_1 > p_2$. For each token in x, a noise token $l \in L$ is inserted with a probability of $1 - p_1$. We hypothesize that the presence of one noise token increases the probability of additional noise tokens. To model this, we use a higher second probability $1 - p_2$ for repeatedly adding an additional noise token. We define an upper bound of $k/2$ for the number of inserted noise tokens c under the assumption that a text snippet with $c \geq k/2$ does not resemble the original language of x anymore. See algorithm 1 for more details.

The algorithm inserting character level noise receives as input a token-level noisified training example x^{Tnoise} and analogous to token-level noise injection, the two thresholds $p_3 \in [0, 1]$ and $p_4 \in [0, 1)$ with $p_3 > p_4$. Additionally, the algorithm receives a character set C, consisting of alphanumeric and punctuation characters from the Latin 1 character set. At each character in x^{Tnoise} character-level noise is injected with a probability of $1-p_3$. The noise consists of either character insertion, omission, or repetition. All three types of noise are equally likely to happen. We hypothesize that the presence of character-level noise makes more such noise likelier. Thus, in case of insertion or repetition, we repeatedly add additional noise characters with a probability of $1 - p_4$. See algorithm 2 for more details.

Our implementation of the approach described in this section using PyTorch will be published at https://github.com/JonathanSchaber/shared_task.

4 Results and Discussion

Our submitted model achieves an F1 score of 96.8% in the official evaluation on the test set, resulting in a second place, 1.4% behind the best model.

Table 2 gives an overview of different hyperparameter settings with the corresponding results on the development and test set.[5] We report the following observations: (1) More fine-grained classes generally lead to better results. (2) There is a strong performance drop from development to test set supporting our noisiness gap assumption. (3) Injecting noise alleviates this drop and, compared to the same configurations without noise, leads to relative performance increases ranging from 1.2% to 2.7% F1 score on the test set.

(4) Increasing embedding dimensionality leads to more stable results over different granularities. (5) Clipping after 100 characters leads to a bisection of training time while on average upholding performance.

Since the test set does not contain languages from A the character-based filter is rarely triggered and its impact on performance is negligible. However, the filter might be important when detecting GSW text in settings where languages in A occur more frequently. More information about hyperparameters and hardware is given in Appendix C.

5 Conclusion

This paper described our submission to the GSWID 2020 shared task. We introduced a BiGRU-based architecture, a character-based filter and a noisifier module. Our evaluation results

[5]The test set evaluation relies on gold labels that were made available after the submission deadline.

show that more fine-grained classes and adding noise to the training data leads to performance increases. Further investigations will concern pre-training, transformer-based architectures, and a more sophisticated noisifier.

Acknowledgments Above all, we would like to thank Simon Clematide who supervised this project and suggested more fine-grained classes. Further, we thank the shared task organizers, especially Pius von Däniken who clarified our questions, and also our proofreaders and reviewers.

References

Joseph Chee Chang and Chu-Cheng Lin. 2014. Recurrent-neural-network for language detection on twitter code-switching corpus. *arXiv preprint arXiv:1412.4314*.

Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. 2014. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Kilian Foth, Arne Köhn, Niels Beuck, and Wolfgang Menzel. 2014. Because size does matter: The hamburg dependency treebank.

Pablo Gamallo, Marcos Garcia, Susana Sotelo, and José Ramom Pichel Campos. 2014. Comparing ranking-based and naive bayes approaches to language detection on tweets. In *TweetLID@ SEPLN*, pages 12–16.

Dirk Goldhahn, Thomas Eckart, and Uwe Quasthoff. 2012. Building large monolingual dictionaries at the leipzig corpora collection: From 100 to 200 languages. In *LREC*, volume 29, pages 31–43.

Ralf Grubenmann, Don Tuggener, Pius Von Däniken, Jan Deriu, and Mark Cieliebak. 2018. SB-CH: A Swiss German Corpus with Sentiment Annotations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*, Miyazaki, Japan. European Language Resources Association (ELRA).

Farheen Hanif, Fouzia Latif, and M Sikandar Hayat Khiyal. 2007. Unicode aided language identification across multiple scripts and heterogeneous data. *Information Technology Journal*, 6(4):534–540.

Nora Hollenstein and Noëmi Aepli. 2014. Compilation of a swiss german dialect corpus and its application to pos tagging. In *Proceedings of the First Workshop on Applying NLP Tools to Similar Languages, Varieties and Dialects*, pages 85–94.

Aaron Jaech, George Mulcaire, Shobhit Hathi, Mari Ostendorf, and Noah A Smith. 2016a. Hierarchical character-word models for language identification. *arXiv preprint arXiv:1608.03030*.

Aaron Jaech, George Mulcaire, Mari Ostendorf, and Noah A Smith. 2016b. A neural model for language identification in code-switched tweets. In *Proceedings of The Second Workshop on Computational Approaches to Code Switching*, pages 60–64.

Tommi Sakari Jauhiainen, Marco Lui, Marcos Zampieri, Timothy Baldwin, and Krister Lindén. 2019. Automatic language identification in texts: A survey. *Journal of Artificial Intelligence Research*, 65:675–782.

David Jurgens, Yulia Tsvetkov, and Dan Jurafsky. 2017. Incorporating dialectal variability for socially equitable language identification. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 51–57.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Tom Kocmi and Ondřej Bojar. 2017. Lanidenn: Multilingual language identification on character window. *arXiv preprint arXiv:1701.03338*.

Rahul Venkatesh Kumar, M Anand Kumar, and KP Soman. 2015. Amritacen_nlp@ fire 2015 language identification for indian languages in social media text. In *FIRE workshops*, pages 26–28.

Yitong Li, Timothy Baldwin, and Trevor Cohn. 2018. What's in a domain? learning domain-robust text representations using adversarial training. *arXiv preprint arXiv:1805.06088*.

Lucy Linder, Michael Jungo, Jean Hennebert, Claudiu Musat, and Andreas Fischer. 2019. Automatic creation of text corpora for low-resource languages from the internet: The case of swiss german. *arXiv preprint arXiv:1912.00159*.

Paul McNamee. 2005. Language identification: a solved problem suitable for undergraduate instruction. *Journal of Computing Sciences in Colleges*, 20(3):94–101.

Jordi Porta. 2014. Twitter language identification using rational kernels and its potential application to sociolinguistics. In *TweetLID@ SEPLN*, pages 17–20.

Younes Samih, Suraj Maharjan, Mohammed Attia, Laura Kallmeyer, and Thamar Solorio. 2016. Multilingual code-switching identification via lstm recurrent neural networks. In *Proceedings of the Second Workshop on Computational Approaches to Code Switching*, pages 50–59.

Arkaitz Zubiaga, Inaki San Vicente, Pablo Gamallo, José Ramom Pichel Campos, Iñaki Alegría Loinaz, Nora Aranberri, Aitzol Ezeiza, and Víctor Fresno-Fernández. 2014. Overview of tweetlid: Tweet language identification at sepln 2014. In *TweetLID@ SEPLN*, pages 1–11.

A Neural Architecture

We formally define our architecture as follows: Let $E(x_i)$ denote a function that returns the embedding for a given character $x_i \in x$. The last hidden states $\overrightarrow{h_n}$, $\overleftarrow{h_0}$ are given by

$$\overrightarrow{h_n}, \overleftarrow{h_0}, \overrightarrow{o_0}, ..., \overrightarrow{o_n}, \overleftarrow{o_n}, ... \overleftarrow{o_0} = \text{BiGRU}(E(x_0), ..., E(x_n)). \quad (1)$$

We feed each last hidden state into a block of dense layers defined as

$$f^{block}(v) = W_2^T * dr(\text{ReLU}(W_1^T * dr(v) + b_1)) + b_2 \quad (2)$$

where $W_1 \in \mathbb{R}^{300\times150}$ and $W_2 \in \mathbb{R}^{150\times50}$ denote the weight matrices of the block's first and second layer, dr denotes a dropout function, b_1 and b_2 denote learnable biases, and ReLU denotes the rectified linear unit activation function.

z_1 is computed as $z_1 = f^{block}(\overleftarrow{h_0})$ and z_2 respectively as $z_2 = f^{block}(\overrightarrow{h_n})$. Note that the weights of the two dense blocks are not shared, but initialized and trained independently. We concatenate z_1 and z_2 to z, which is then fed through a final layer formalized as follows:

$$f^{final}(v) = \text{log-softmax}(W^T * v + b) \quad (3)$$

with $W \in \mathbb{R}^{100\times Q}$ where Q is the `# target classes`.

B Noisifier

As examples for data containing token- and character-level noise, consider the following two made up text snippets.[6] Character sequences we regard as noise are boldfaced.

> Dä bus isch stablibe, mis ticket nüme gültig **trying to stay chill**
>
> **ooohhhh neiiiii** mir hänns nöd gschafft

In the following algorithms, $r()$ denotes a function which returns a random value $\in [0, 1]$.

In algorithm 2 parameter A contains {'omission', 'insertion', 'repetition'}.

For a given example input our noisifier with parameters settings as shown in the hyper parameter table in Appendix C introduces noise structures into non-noisy texts, like the following:

[6]For copyright reasons we do not cite or display real tweets in the publication.

Algorithm 1 Token-level noise injection

Input: x, p_1, p_2, L
Output: x^{Tnoise}
t ← split x into tokens
initialize array u
for each $t_j \in t$ **do**
  if $r_1 \leftarrow r() > p_1$ **then**
    add randomly chosen token $l \in L$ to u
    while $r_2 \leftarrow r() > p_2 \wedge c < k/2$ **do**
      add randomly chosen token $l \in L$ to u
    end while
  end if
  append t_j to u
end for
return concatenate u to string x^{Tnoise}

Algorithm 2 Character-level noise injection

Input: $x^{Tnoise}, p_3, p_4, C, A$
Output: x^{Cnoise}
initialize empty string x^{Cnoise}
for each $x_i \in x^{Tnoise}$ **do**
  if $r_3 \leftarrow r() > p_3$ **then**
    $a \leftarrow$ choose random action $\in A$
    if $a =$ 'omission' **then**
      continue
    else if $a =$ 'insertion' **then**
      $b \leftarrow$ choose random character $\in C$
    else if $a =$ 'repetition' **then**
      $b \leftarrow x_i$
    end if
    add b to x^{Cnoise}
    while $r_4 \leftarrow r() > p_4$ **do**
      add b to x^{Cnoise}
    end while
  end if
  add x_i to x^{Cnoise}
end for
return x^{Cnoise}

clean: Viele Personen sind nicht der Überzeugung.
noisy: Viele Personen sind nicht der **Üerzeugunnng**.

clean: Hast du schon die neue xbox 3 gesehen?
noisy: Hast du **music** schon die neue xbox 3 **geesehen**?

clean: You'll never guess what happened this morning.
noisy: You'll never guess **Jwhat** happened this **morninnng**.

clean: Le tigre est un grand chat de proie originaire d'Asie.
noisy: Le tigre **estt** un grand chatde proie originaire d'Asie.

clean: C'è ancora una mancanza di chiarezza, non possiamo farci nulla.
noisy: C'è ancor una mancanza di chiarezza, non possiamo **St. Moritz Frisör** farci nulla.

As is obvious from these examples, the noise injected by the noisifier still looks quite different from human created noise, thus a more sophisticated noisifier is desirable.

C Configurations

Table 2 only lists parameters which are changed during ablation testing. In the table below, we report the parameters we left unchanged during ablation testing.

Parameter	Value
hidden size h	300
dense-block layer-in size	300
dense-block layer-betw. size	150
dense-block layer-out size	50
final-block layer-in size	100
final-block layer-out size	`# target classes`
dropout	0.1
learning rate	0.001
number of epochs	15
p_1	0.99
p_2	0.6
p_3	0.97
p_4	0.5
character-filter threshold	0.8

Table 3: Hyperparameters maintained constant during all experiments.

After two epochs the learning rate is decreased from 0.001 to 0.0001 and after six epochs the learning rate is further decreased to 0.00003.

We ran our models on a NVIDIA GeForce GTX TITAN X graphics processing unit.