

Figure 1: Character confusion set. Restricted to lower case English alphabet for visualisation. Character pairs which are close to each other on a keyboard tend to have higher values. Values in each row sum up to 1.

Normalising typo positions for all `<typo_string, correct_string>` pairs allows us to compute a probability of making a typing mistake by each of 100 percentiles (e.g. probability at 66th percentile corresponds to a probability of making a typo in first 2/3 of the string). Based on an input string length we convert probabilities for 100 percentiles to a probability mass function over all character positions in a string. With typo probabilities assigned to each individual character in an input string, it is trivial to iterate over such string and generate typos, following the patterns exhibited by humans.

We find that typos tend to happen closer to the end of the string, confirming findings in earlier studies (Mitton, 1996). The distribution of normalised typo positions is presented in Figure 2.

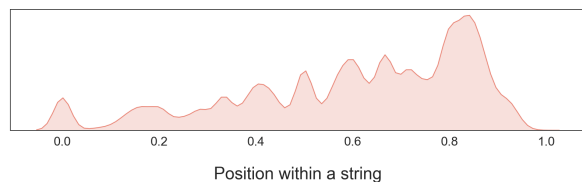


Figure 2: Distribution of a normalised typo position within a string.

3.3 Typo Generation

Using the statistics described above, we are able to generate any number of realistic typos that closely mimic human errors. Our algorithm accepts any string as an input and generates a realistic-looking

typo, based on string length and statistics described above. We run this algorithm directly on search logs, attempting typo generation for each record. On average we introduce 1 typo per record, but due to the stochastic nature of the typo generation procedure, some records may have 0 or 2 typos.

Using this procedure we generate a dataset of 94M examples, which is significantly larger than the labeled records we could obtain with any other method. A byte pair encoding (BPE) tokenizer (Sennrich et al., 2015) is then fit on this dataset, resulting in a vocabulary of 12K tokens. Similar to Zhou et al. (2017), we find subword tokenization to be the best way of representing model inputs. The whole dataset is tokenized using the BPE tokenizer, shuffled, and split in 100:1:1 proportions into train, validation, and test sets. This approach allows us to have a training set that is very similar to the data seen in production, thereby minimising possible distribution drifts.

One side effect of constructing `<typo_string, correct_string>` pairs directly from search logs is that noise is introduced over already erroneous queries, like gibberish, niche or partial searches. This may appear detrimental to model performance, however, surprisingly, we observe that in practice noise introduced over erroneous queries does not hurt the quality of model outputs. Instead, given the large size and diversity of the dataset, it forces the model to output sequences identical to the input sequence by default, and to only attempt correction in cases of high certainty (e.g. if a typo is made in a sufficiently popular token). By forcing model outputs to be identical to model inputs in case of gibberish queries, this setup effectively addresses the overcorrection problem (Movin, 2018; Zhu et al., 2019).

Additionally, as some input tokens are compounds (e.g. email addresses, containing first and last names, domain address, etc), this setup forces the model to handle multiple typos in several distinct entities within a single contiguous string.

The ability to train a well performing model directly on the noise generated over unprocessed search logs is surprising, and to the best of our knowledge was not demonstrated before.

4 Spelling Correction Model

We train a denoising autoencoder transformer model to recover the original query (without noise)

from the query which potentially contains a typo. As model inputs and labels are generated directly from logs data, the distribution of queries is similar between training and inference settings, thereby minimising distribution drifts and biases. Additionally, as queries are seen by the model according to their popularity, the model will naturally learn most frequent terms and will be forced to learn to ignore (and not correct typos on) infrequent, often erroneous and incomplete queries. The production version of our model is a transformer with 4 layers, 2 attention heads, hidden layer size of 256, trained for 1.6M steps with default learning rate warm-up and decay. This results in a model with 10M trainable parameters. For model implementation we rely on a `tensor2tensor`⁴ library (Vaswani et al., 2018) and use the hyper-parameter set defined in the library as `transformer_small`).

5 Experiments

Below we present two experiments: one comparing approaches to artificial noise generation, and another demonstrating ability to perform transfer to other languages for which no labels are available. Maximising model quality was not our goal in these experiments, and we expect that additional tuning of the model, vocabulary generation and training procedures, as well as using beam search score as a confidence threshold will yield significant improvements in quality of spelling correction.

5.1 Realistic vs Uniformly Generated Typos

We compare two approaches of generating training data: using realistic typos (*Real*) and using a baseline (*Base*) approach which generates typos in a uniformly random way. For *Real* approach we use typo statistics derived from search logs and for *Base* approach all typo types and string positions are treated as equally probable, and characters for *Insertion* and *Substitution* are chosen uniformly at random from a set of lowercase and uppercase English alphabet letters. For this comparison we train identical denoising transformer models on artificially generated typos for two datasets: HubSpot search logs (94M original search queries, not tokenized) and a dataset of 3036 Project Gutenberg books (tokenized into 51M tokens, 989K unique) (Lahiri, 2014). For each dataset we generate both uniform and realistic versions of

a typo for exactly the same set of input strings. Apart from tokenization for the Gutenberg dataset, no data preprocessing is performed. Models trained on the Gutenberg dataset are evaluated on ground truth datasets of English typos: Wikipedia Common Misspellings⁵, and Aspell, Birkbeck, Holbrook datasets⁶. Models trained on HubSpot search logs are evaluated on a dataset of 195K `<typo_string, correct_string>` pairs described in section §3.1. All models are identical to the one described in section §4 and are trained for 200K steps (5-6 hours on Nvidia V100 GPU). We report sequence-level accuracy on both `<typo_string, correct_string>` (*Typos*) and `<correct_string, correct_string>` (*Identity*) pairs. Accuracy on *Identity* pairs is equivalent to $1 - FPR$, where *FPR* is False Positive Rate⁷. Results of this experiment are presented in Table 2.

Dataset	Typos		Identity	
	Real	Base	Real	Base
Search Typos	56.84	43.70	96.09	96.83
Wikipedia	65.92	63.58	84.90	86.39
Aspell	40.30	37.66	83.78	84.22
Birkbeck	33.34	29.27	85.14	85.40
Holbrook	17.92	17.25	73.92	74.92

Table 2: Comparison of realistic and uniformly random typo generation approaches.

Experiment results suggest that there is a considerable benefit in generating typos in a realistic manner, which is especially evident in the case of our in-house search typos dataset, from which human error patterns were derived. The fact that error patterns derived from search typos may be successfully transferred to other domains (like Wikipedia, Aspell, and Birkbeck datasets) shows that we are able to at least partially capture fundamental (and not dataset-specific) statistics about human spelling mistakes. In the next section we challenge this conclusion further, attempting to apply our method in non-English domains where no labeled data is available.

⁵https://en.wikipedia.org/wiki/Wikipedia:Lists_of_common_misspellings/For_machines

⁶<https://www.dcs.bbk.ac.uk/~ROGER/corpora.html>

⁷https://en.wikipedia.org/wiki/False_positive_rate

⁴<https://github.com/tensorflow/tensor2tensor>

5.2 Transfer to Resource-Scarce Languages

Our procedure of training data generation is based on introduction of noise to natural language and relies solely on pre-computed typo-related statistics. Under a bold assumption that such statistics are largely language-agnostic, we show that it is possible to train a denoising transformer-based spelling correction model in settings where no `<typo_string, correct_string>` pairs are available. Leaving other statistics unchanged, we convert the character confusion matrix from English language to a target language using a QWERTY keyboard mapping. This way each English character is mapped to a character on the same keyboard key used in a target language layout. Using updated statistics, we train simple models for Russian⁸, Arabic (Aly and Atiya, 2013), Greek⁹, and Setswana¹⁰ languages. Datasets for Arabic, Greek, and Setswana are split into individual tokens. Datasets for Greek and Russian are lowercased. We use the same model configuration and training procedure as in section §5.1. In Table 3 we report the number of unique examples and tokens for each dataset, alongside with sequence-level accuracy on a test set (not seen by BPE tokenizer and the model during training).

Dataset	Example #	Token #	Accuracy
Arabic	4,096,407	318,521	83.33
Greek	9,491,753	270,269	93.97
Russian	2,679,222	324,867	91.83
Setswana	2,348,161	61,382	94.48

Table 3: Language transfer results.

Results indicate that this simple approach proves itself useful for bootstrapping spelling correction systems in settings when no labels are available. These findings may be especially helpful for languages suffering from scarcity of available resources, such as the majority of languages in Africa (Martinus and Abbott, 2019).

⁸https://github.com/Koziev/NLP_Datasets/blob/master/Samples/prep%2Bnoun.zip

⁹<https://repositori.upf.edu/handle/10230/19963>

¹⁰<https://repo.sadilar.org/handle/20.500.12185/404>

6 Production Usage

Trained model is loaded in memory using the TensorFlow (Abadi et al., 2016) SavedModel format, and is fed all input strings shorter than `MAX_INPUT_LENGTH=20`. We limit max input size in order to ignore abnormally long inputs and to provide latency guarantees, as transformer time complexity is quadratic in the input sequence length.

Beam search of size 2 is performed when selecting top output sequence candidate, and we find that increasing beam size gives only minimal quality improvements at the expense of significantly higher latency. Beam search score is treated as a confidence score and is assigned to every prediction. Empirically chosen cut-off of 0.5 is used for serving predictions (i.e. all spelling corrections with score below this threshold are ignored), resulting in 1.5% of queries being corrected. Relationship between confidence threshold and spelling correction rate on HubSpot search logs is presented in Figure 3.

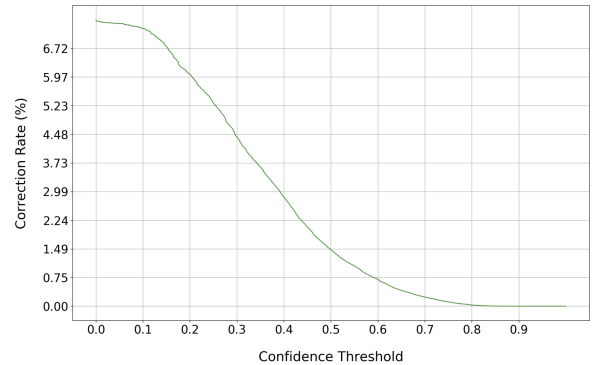


Figure 3: Correction Rate vs Confidence Threshold.

7 Future Work

This paper presents our first iteration on deep learning spelling correction, with multiple avenues for further improvement and research. In particular, we leave several items for future work:

- **Model architecture improvements.** Currently, we use a default transformer implementation, and there may be benefit in increasing model capacity, vocabulary and beam search size, using custom architectures, as well as a combination of models suggested by Li et al. (2018). Additional techniques like label smoothing, checkpoint averaging,