Informationsverarbeitung I (Converational AI) WS 2020
Prof. Christoph Ringlstetter

Grapheme to Phoneme Conversion for German

Rafael Kupsa
Matrikelnr.: 12256980

March 8, 2021

Contents

1 Introduction

This short article is an attachment to the coding project I realized for the seminar "Informationsverarbeitung I" (Conversational AI) at the LMU university of Munich which I attended as part of my Master's degree in Computational Linguistics. The project's goal was to implement a python module capable of converting German language text to a phonological representation which is also called grapheme to phoneme conversion or G2P for short. This module could then theoretically be used as part of a speech synthesis pipeline in a larger conversational AI.

The G2P conversion works in the following way: The text is normalized and its individual words are looked up in a large pronunciation dictionary which was constructed specifically for this project from pronunciation data taken from Wiktionary (`https://en.wiktionary.org`, see section 2). If the dictionary does not contain the word it is fed to a recurrent neural network (see section 3) that has been trained on

said dictionary which will return a prediction for the word's pronunciation. The module allows the use of two different RNN models for comparison. Both use an encoder-decoder architecture with LSTM layers, one uses one-hot representations of words and pronunciations as input, the other uses extra layers to convert the words and pronunciations to embeddings. As will be shown in the article, the model using embedding layers performs much better.

The module as wall as the scripts that have been used to create and normalize the dictionary and to train both models can be found at `https://github.com/RafaelKupsa/g2p` as well as a README file on how to use the module.

2 The dictionary

The dictionary used for training the models as well as looking up words during G2P conversion is constructed from a web crawling script (create_dictionary.py) that extracts words and their IPA pronunciation from Wiktionary and can be inspected at (pronunciation_dictionary.txt). The format is based on the CMU Pronouncing dictionary (`http://www.speech.cs.cmu.edu/cgi-bin/cmudict`) for the English language but uses an adapted notation system for the phonemic spelling (abbreviated to CNS - custom notation system, in this article). This format is frequently used in G2P conversion models such as the python module g2p-en (`https://pypi.org/project/g2p-en/`). The dictionary is essentially a long list of words followed by the indivual phonemes of its pronunciation seperated by space characters (see table 1). An explanation for the phoneme notation system used is provided in table 2.

```
...
fünfzigtägig F UEH N F TS IH C T AEE G IH C
fünfzigwöchig F UEH N F TS I C V OEH C IH C
fungibilität F UH NG G I B I L I T AEE T
fungieren F UH NG G II R EX N
funk F A NG K
...
```

Table 1: Format of the pronunciation dictionary

The web crawling script works by going through every Wiktionary page containing German terms with IPA pronunciations which Wiktionary conveniently provides a list of at `https://en.wiktionary.org/wiki/Category:German_terms_with_IPA_pronunciation`. Every German term is then written to one line of a text file followed by its pronunciation converted from IPA to my custom phoneme notation. Words containing number symbols

IPA	CPN	Example	IPA	CPN	Example
a/ɑ	A	K*a*t*a*strophe	b	B	*B*i*b*er
aː/ɑː	AA	V*a*se	ç	C	Ei*ch*e
e	E	*e*lektrisch	d	D	*d*och
eː	E	l*e*sen	dʒ	JH	*Dsch*ungel
ɛ	EH	B*e*tt	f	F	*f*ün*f*
ɛː	AEE	Universit*ä*t	ɡ	G	*G*asse
i	I	m*i*n*i*mal	h	H	*H*aus
iː	II	l*ie*ben	j	J	*j*agen
ɪ	IH	b*i*tten	k	K	*K*e*k*se
o	O	M*o*no*o*log	l	L	spie*l*en
oː	OO	Mono*o*g	m	M	Bau*m*
ɔ	OH	T*o*nne	n	N	Ha*n*d
ø	OE	m*ö*bliert	ŋ	NG	si*ng*en
øː	OEE	l*ö*blich	p	P	*P*a*pp*el
œ	OEH	*ö*ffnen	p͡f	PF	To*pf*
u	U	M*u*sik	r/ʀ/ʁ	R	*r*iesig
uː	UU	T*u*be	s	S	Fau*s*t
ʊ	UH	K*u*ppel	ʃ	SH	Ta*sch*e
y	UE	Ph*y*sik	t	T	*T*in*t*e
yː	UEE	Gem*ü*se	t͡s	TS	Ka*tz*e
ʏ	UEH	R*ü*stung	t͡ʃ	CH	Qua*tsch*
aɪ̯	AI	b*ei*ßen	v	V	*W*olke
aʊ̯	AU	t*au*send	x	X	ma*ch*en
ɔʏ̯/ɔɪ̯	OI	k*äu*flich	z	Z	*s*au*s*en
ɐ	AX	Mutt*er*	ʒ	ZH	Gara*g*e
ə	EX	b*e*gegn*e*n	ʔ	?	ver-achten

Table 2: The custom phonemic notation system used in this project

are not added to the dictionary and word stress was disregarded altogether (see section 4).

Since Wiktionary's IPA transcription system is not completely consistent, a second script to "clean up" the dictionary (clean_up_dictionary.py) was used to convert the extracted dictionary to a more consistent form (pronunciation_dictionary_clean.txt) using regular expressions.

List of adjustments:

- Words are converted to lower case and if they occur more than once after that, only one entry is being kept. (Those are usually verbs and nouns with the same form which have the same pronunciation anyway.)
- All characters not in the standard German alphabet (a-z, ä, ö, ü and ßhave been replaced by "_" (e.g. ",", "-", "é", ...)

- The glottal stop (IPA: /ʔ/, CPN: ?) at the beginning of words has been deleted in words where it was present since it is inconsistently used in Wiktionary and can be inferred from the vowel at the beginning of a word.
- The pronunciation of the r-sound after vowels and before consonants or the end of the word is inconsistent in Wiktionary (occuring as /r/, /ɾ/, /ʁ/, /ʁ̞/ or /ɐ̯/) and has been regularized to a-schwa (IPA: /ɐ̯/, CPN: AX)
- The pronunciation of the optional schwa in morphemes ending in -en is inconsistent in Wiktionary (occuring as /ən/, /n̩/, /m̩/ or /ŋ̩/depending on the preceding consonant) and has been regularized to include the schwa everywhere (IPA: /ən/, CPN: EX N)
- The pronunciation of the optional schwa in morphemes ending in -el is inconsistent in Wiktionary (occuring as /əl/ or /l̩/) and has been regularized to include the schwa everywhere (IPA: /əl/, CPN: EX L)

After creating and cleaning up the dictionary it consists of 35,307 entries. Some slight errors and incosistencies are still present but they should not affect the subsequent training of the RNN too much.

Their are other pronunciation dictionaries that exist for German language already which could be used to train an RNN but I chose to create my own dictionary as a challenge for this project.

3 The RNN model

The dictionary is far from complete, in fact due to the compounding nature of the German language it cannot possibly contain every single word and its pronunciation. It also does not contain most proper names nor every paradigm of every inflecting German word ("machen", "mache", "machst", "macht", ...). While it would be possible to approximate the pronunciation of unknown words by analyzing their morphology and having seperate pronunciation rules for inflectional or derivational morphemes, another approach is to use a recurrent neural network that is trained on the dictionary and tries to predict an unknown word's pronunciation.

For this project I used the Python Tensorflow library (`https://pypi.org/project/tensorflow/`) and in particular the included Keras API which offers convenient ways to model, adapt, train and test neural networks. I built two similar models to compare two approaches to the problem. Both models are sequence-to-sequence models since we want to predict a sequence of phonemes from a sequence of graphemes and an encoder-decoder architecture with LSTM layers. Since there is no one-to-one correspondence between grapheme and phoneme, an encoder-decoder model is a good approach. Additionally, LSTMs are perfect for learning order dependences in sequences and the G2P problem has a lot of those, e.g. "-att-" → "A T" vs. "-at-" → "AA T" (vowels followed by double consonants are usually short).

Both models were trained with a maximum of 150 epochs, however, they would stop early if no improvement in validation loss was calculated for the last 5 epochs.

The first model uses a simple one-hot representation of the words and pronunciations which are then fed into the LSTM encoder and decoder respectively. The model was trained numerous times with differing hyperparameters: hidden nodes from 128 to 256 in the LSTM layers, learning rates from 0.0003 to 0.01. The best results were achieved with 256 hidden nodes and a small learning rate of 0.0003 with the model training for 149 epochs before stopping. The training loss at the end was about 0.29 and the validation loss 0.36 while the training accuracy ended at approximately 0.28 and the validation accuracy at 0.27 (see figure 1). A random sample of 100 words in the test corpus however achieved an accuracy of 0.43 and especially shorted words tended to be fairly accurate.

The second model uses embedding layers for both encoder and decoder with an embedding size of 256 and several dropout layers to account for overfitting from such a large number of nodes. The LSTM's hidden nodes and the learning rate were again 256 and 0.0003 respectively and two different dropout rates were tried out (0.5 and 0.3). The best results were achieved with a dropout rate of 0.3 for which the model trained for 83 epochs before stopping. At the end both training and test loss were around 0.08 and both training and testing accuracy around 0.85 (again, see figure 1). A random sample of 100 words in the test corpus achievedan accuracy of 0.77.

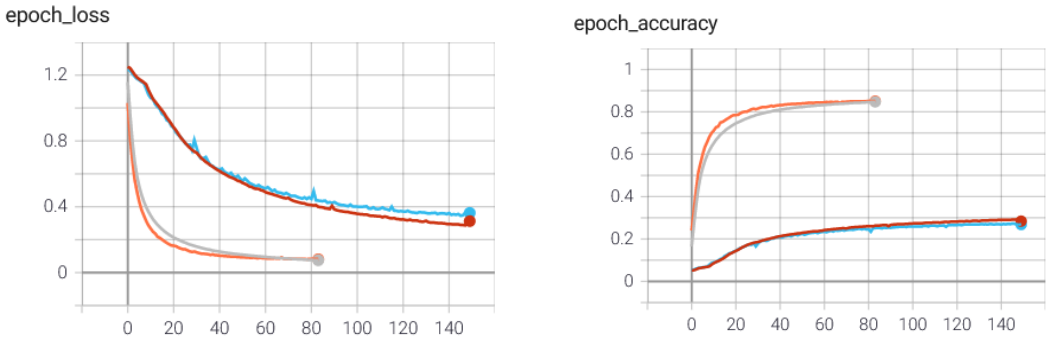

Figure 1: Tensorboard graphs: Progress of loss and accuracy per epoch

As immediately becomes obvious, the second model performs much, much better. Learning embeddings for both the words and their pronunciations proved to be well worth the effort and lifted the end result from a nice try to actually usable. A look at the mistakes the two models made in the test samples (see figure 2) shows that the first model really struggles with longer words in general, most of the times the beginning of the predicted pronunciation is correct while the phonemes at the end seem to be mostly guesswork, e.g. "hochaktuell" was predicted as "H OO X ? A K T L UU N" while the correct form would have been "H OO X ? A K T U EH L".

The second model on the other hand seems to really have an idea about how to predict the whole word with most mistakes having understandable causes, e.g. predicting the "i" in "iodidhaltig" as an "I" instead of "J" or falsely guessing which vowel is long in

"analysis".

```
jurist --> J U R IH S T -- prediction: J U R IH S T -- correct? True
abbrechen --> A P B R EH C EX N -- prediction: A P B R EH NG EX N -- correct? False
rot_gelb --> R OO T G EH L P -- prediction: R OO T G EE B EX -- correct? False
hochaktuell --> H OO X ? A K T U EH L -- prediction: H OO X ? A K T L UU N -- correct? False
frikadelle --> F R IH K A D EH L EX -- prediction: F R I K A D EX L EX N -- correct? False
blaublütig --> B L AU B L UEE T IH C -- prediction: B L AU B L UEE T IH C -- correct? True
iodidhaltig --> J OO D II T H A L T IH C -- prediction: I OO D I T AA L T IH C -- correct? False
salzstange --> Z A L TS SH T A NG EX -- prediction: Z A L TS SH T A NG EX -- correct? True
sauerampfer --> Z AU AX ? A M PF AX -- prediction: Z AU AX ? A M PF AX -- correct? True
analysis --> A N AA L UE Z IH S -- prediction: A N A L UEE Z IH S -- correct? False
schachfigur --> SH A X F I G UU AX -- prediction: SH A X F I G UU AX -- correct? True
siamesisch --> Z I A M E Z IH SH -- prediction: Z I A M E Z IH SH -- correct? True
```

Figure 2: Excerpt from a random test sample (model 1 top, model 2 bottom)

The embedding layers seem to make a huge difference and this is not surprising. Similar graphemes and phonemes should have similar representations in the end which should lead to generalization of their features. For example, without embeddings it might have been learned that after the phoneme "B" there could be an "A" or an "R" but probably not an "M". With embeddings this property of "B" might have been learned to also apply to other sounds such as "P", "G", "K", "D" or "T" and this property of "A" might have been learned to also apply to other vowels, etc. So for everything the model learns from a new word containing "B" it also learns something about "P", "G" and so on which multiplies again and again.

4 Conclusion and ideas for improvement

An accuracy score of 85% is not the worst for such a task especially since tiny mistakes in pronunciation would may not even be noticed or affect comprehension if the predicted pronunciations were used as input for an actual speech synthesizer. However I imagine the score could be even higher with some improvements to both the data and the model.

Firstly, the data could probably be improved by considering stress in the pronunciations. Stress was left out here for simplicity but it could actually help improving the accuracy since many vowel qualities depend on stress patterns, e.g. schwa and a-schwa are never stressed. A speech synthesizer would also need some indication on where to stress syllables to sound naturally.

Secondly, the data could have more words to learn from. The CMU Pronouncing Dictionary for English has around 134,000 words and this is considering English has almost no inflection which would extend the dictionary even more. In this project's dictionary most verbs are in the infinitive form and most nouns are in the nominative singular form which probably results in a bias to those forms and lower accuracy when predicting inflected forms. Unfortunately, Wiktionary does not provide pronunciations for most inflected forms, so a better source for the data would probably improve the data.

While the model could almost certainly be improved by further adjusting and tweaking some hyperparameters, it would probably be even more beneficial to make some changes in the overall architecture of the model and maybe try a model using a bidirectional encoder or an attention mechanism or both. As seen from the test sample results for the first model, the model has difficulties at the end of words and a bidirectional encoder could help even this bias out. A lot of phonemes also depend on the ones coming immediately after, e.g. in "Auslautverh'artung". An attention model could help with stress and vowel length related pronunciation phenomena, e.g. the fact that there probably isn't another long vowel in a word if their has already been one (except in compounds).

In conclusion, for quick approximation the second model using embedding layers in combination with looking up known words in the pronunciation dictionary should suffice while the first model does not bring adequate results. However, if more precision and nuance are needed, there is still room for improvement both in the data and the model architecture. Additionally, skipping a step and directly learning sound waves from words would also be an option if actual spoken output is the end goal.

5 References

Chollet, F. (2017, September 29). A ten-minute introduction to sequence-to-sequence learning in Keras. *The Keras Blog.* `https://blog.keras.io/a-ten-minute-introduction-to-sequence-to-sequence-learning-in-keras.html`

Chollet, F. (2017, September 29). Character-level recurrent sequence-to-sequence model. *Keras.* `https://keras.io/examples/nlp/lstm_seq2seq/`

Epp, R. (2021). Predicting English Pronunciations. An in-depth grapheme-to-phoneme conversion tutorial using Keras. *Ryan Epp Blog.* `https://www.ryanepp.com/blog/predicting-english-pronunciations#Baseline-Model`