applied sciences

Type of the Paper (Article, Review, Communication, etc.)

Grapheme-to-Phoneme Conversion with Convolutional Neural Networks

Sevinj Yolchuyeva[*1], **Géza Németh**, **Bálint Gyires-Tóth**

{syolchuyeva, nemeth, toth.b}@tmit.bme.hu

[*]Department of Telecommunications and Media Informatics, University of Budapest Technology and Economics, 1111, Budapest, Hungary

Abstract: Grapheme-to-phoneme (G2P) conversion is the process of generating pronunciation for words based on their written form. It has a highly essential role for natural language processing, text-to-speech synthesis and automatic speech recognition systems. In this paper, we investigate convolutional neural networks (CNN) for G2P conversion. We propose a novel CNN-based sequence-to-sequence (seq2seq) architecture for G2P conversion. Our approach includes an end-to-end CNN G2P conversion with residual connections, furthermore, a model, which utilizes a convolutional neural network (with and without residual connections) as encoder and Bi-LSTM as a decoder. We compare our approach with state-of-the-art methods, including Encoder-Decoder LSTM and Encoder-Decoder Bi-LSTM. Training and inference times, phoneme and word error rates were evaluated on the public CMUDict dataset for US English, and the best performing convolutional neural network based architecture was also evaluated on the NetTalk dataset. Our method approaches the accuracy of previous state-of-the-art results in terms of phoneme error rate.

Keywords: Grapheme-to-Phoneme (G2P), encoder-decoder; LSTM; 1D convolution; Bi-LSTM; Residual Architecture

1.Introduction

The process of grapheme-to-phoneme (G2P) conversion generates the phonetic transcription from the written form of words. The spelling of the word is called grapheme sequence (or graphemes), the phonetic form is called phoneme sequence (or phonemes). It is essential to develop a phonemic lexicon in text-to-speech (TTS) and automatic speech recognition (ASR) systems. For this purpose, G2P techniques are used, and getting state-of-the-art performance in these systems depends on the accuracy of G2P conversion. For instance, in ASR acoustic models, the pronunciation lexicons and language models are critical components. Acoustic and language models are built automatically from large corpora. Pronunciation lexicons are the middle layer between acoustic and language models. For a new speech recognition task, the performance of the overall system depends on the quality of the pronunciation component. In other words, the system's performance depends on G2P accuracy. For example, the G2P conversion of word 'speaker' is 'S P IY K ER'. In TTS systems a high-quality G2P model is also an essential part and has a great influence on the overall quality. Inaccurate G2P conversion results in unnatural pronunciation or even incomprehensible synthetic speech.

[1] Corresponding author:

syolchuyeva@tmit.bme.hu

2. Previous Works

G2P conversion has been studied for a long time. Rule-based G2P systems use a wide set of grapheme-to-phoneme rules [1, 2]. Developing such a G2P system requires linguistic expertise. Additionally, some languages (such as Chinese and Japanese) have complex writing systems, and building the rules is labor-intensive and it is extremely difficult to cover most possible situations. Furthermore, these systems are sensitive to out of vocabulary (OOV) events. Other previous solutions used joint sequence models [3, 4]. These models create an initial grapheme-phoneme sequence alignment, and by using this alignment, it calculates a joint n-gram language model over sequences. The method proposed by [3] is implemented in the publicly available tool Sequitur[2]. In one-to-one alignment, each grapheme corresponds only to one phoneme and vice versa. An "empty" symbol is introduced to match grapheme and phoneme sequences. For example, the grapheme sequence of 'CAKE' matches the phoneme sequence of 'K EY K', and one-to-one alignment of these sequences is C → K, A → EY, K → K, and the last grapheme 'E' matches the "empty" symbol. Conditional and joint maximum entropy models use this approach [5]. Later, Hidden Conditional Random Field (HCRF) models were introduced in which the alignment between grapheme and phoneme sequence is modelled with hidden variables [6, 7]. The HCRF models usually lead to very competitive results, however, the training of such models is very memory and computationally intensive. A further approach utilizes conditional random fields (CRF) and Segmentation/Tagging models (such as linear finite-state automata or transducers, FSTs), then use them in two different compositions [8]. The first composition is a joint-multigram combined with CRF; the second one is a joint-multigram combined with Segmentation/Tagging. The first approach achieved 5.5% phoneme error rate (PER) on CMUDict.

Recently, neural networks have been applied for G2P conversion. Neural network based G2P conversion is robust against spelling mistakes and OOV words; it generalizes well. Also, it can be seamlessly integrated into end-to-end TTS/ASR systems (that are constructed entirely of deep neural networks) [15]. In this paper, a TTS system (Deep Voice) is presented which was constructed entirely from deep neural networks. Deep Voice lays the groundwork for truly end-to-end neural speech synthesis. Thus, the G2P model is jointly trained with further essential parts of the speech synthesizer and recognizer, which increase the overall quality of the system.

LSTM has shown competitive performance in various fields, like acoustic modelling [9] and language understanding [10]. One of the early neural approaches investigates unidirectional Long Short-Term Memory (ULSTM) with full output delays, which achieved 9.1% phoneme error rate [11]. In the same paper, a deep bidirectional LSTM (DBLSTM) was combined with connectionist temporal classification (CTC) and joint n-gram models for better accuracy (21.3% word error rate). Note that CTC objective function was introduced to infer speech-label alignments automatically without any intermediate process, leading to an end-to-end approach for ASR [44]. CTC technique has combined with CNN, LSTM for the various speech-related tasks [45].

Due to utilizing an encoder-decoder approach for the G2P task, a separate alignment between grapheme sequences and phoneme sequences became unnecessary [12, 13].

Alignment based models of unidirectional LSTM with one layer and bi-directional LSTM (Bi-LSTM) with one, two and three layers were also previously investigated [13]. In this work, alignment was explicitly modelled in the G2P conversion process by the context of the grapheme. A further work, which applies deep bi-directional LSTM with hyperparameter optimization (including the number of hidden layers, optional linear projection layers, optional splicing window at the input) considers various alignment schemes [14]. The best model with hyperparameter optimization achieved 5.37% phoneme (PER)and 23.23% word error rate (WER). Multi-layer bidirectional encoder with gated recurrent units (GRU) and deep unidirectional GRU as a decoder achieved 5.8% PER and 28.7% WER on CMUDict [15].

[2] https://www-i6.informatik.rwth-aachen.de/web/Software/g2p.html, Access date: 9th August 2018

Convolutional neural networks have achieved superior performance compared to previous methods in large-scale image recognition [16, 17]. Recently, these architectures were also applied to Natural Language Processing (NLP) tasks, including sentence classifications and neural machine translation. Nowadays, completely convolutional neural networks may achieve superior results compared to recurrent solutions [18, 19].

Sequence-to-sequence (seq2seq) learning, or encoder-decoder type neural networks have achieved remarkable success in various tasks, such as speech recognition, text-to-speech synthesis, machine translation [20, 21, 22, 42]. This type of network is used for several tasks, and its performance has also been enhanced with attention mechanism [19, 42, 43]. In this structure, the encoder computes a representation of each input sequence, and the decoder generates an output sequence based on the learned representation. In [43], bidirectional multi-layer recurrent neural network based seq2seq learning was investigated in two architectures: a single Bi-LSTM/Bidirectional Gated Recurrent Unit (Bi-GRU) layer and two Bi-LSTM/Bi-GRU layers. Both Bi-LSTM and Bi-GRU uses both past and future contexts. Moreover, bidirectional decoder was proposed for neural machine translation (NMT) in [46]. Both encoder and decoder are Bi-GRU, but this model is applicable to other RNNs, such as LSTM. By introducing a backward decoder, the purpose of which is to exploit reverse target-side contexts, the results of NMT task was improved. For speech recognition, several sequence-to-sequence models including connectionist temporal classification (CTC), the recurrent neural network (RNN) transducer, an attention-based model [51] have been analyzed. The basics of sequence modelling with convolutional networks are summarized in [52]. Furthermore, the key components of the temporal convolution network (TCN) have also been introduced and some vital advantages, and disadvantages of using TCN for sequence predictions instead of RNNs were analyzed as well.

The encoder-decoder structure was studied for the G2P task [13, 15, 23] before, but usually, LSTM and GRU networks were involved. For example, Baidu's end-to-end text-to-speech synthesizer, called Deep Voice, uses the multi-layer bidirectional encoder with GRU's non-linearity and an equally deep unidirectional GRU decoder [15]. Until now the best result for G2P conversion was introduced by [23], which applied an attention-enabled encoder-decoder model and achieved 4.69% PER and 20.24% WER on CMUDict. Furthermore, G2P-seq2seq[3] is based on neural networks implemented in the TensorFlow framework with 20.6% WER.

According to our knowledge, our approach is the first that uses convolutional neural networks for G2P conversion. In this paper, we present one general sequence-to-sequence and four encoder-decoder models. These are introduced in Section 3. Our goal was to achieve and surpass (if possible) the accuracy of previous models and to reduce the training times (which is quite high in case of LSTM/GRU).

The remaining parts of this paper are structured as follows: Section 3 discusses the possibility to apply convolutional neural networks for sequence-to-sequence based grapheme-to-phoneme conversion. Datasets, training processes, and evaluation of the proposed models are presented in Section 4. Section 5 analyzes the results of the models, and finally, the conclusion is drawn in Section 6.

3. Convolutional Neural Networks for Grapheme to Phoneme Conversion

Convolutional neural networks are used in various fields, including image [24, 25], object [16, 26, 27] and handwriting recognition [27, 28], face verification [29], natural language processing [30, 31] and machine translation [19]. The architecture of an ordinary CNN is composed of many layer types (such as the convolutional layers, pooling layers, fully connecting layers, etc.) where each layer carries out a specific function. The convolutional and pooling layers are for representation learning, while the fully connected layers on the top of the network are for modelling a classification or regression problem. One of the main reasons that make convolutional neural networks superior to previous methods is that CNNs perform representation learning and modelling jointly, thus a quasi-

[3] https://github.com/cmusphinx/g2p-seq2seq, Access date: 9th August 2018

optimal representation is extracted from the input data for the machine learning model. Weight sharing in the convolutional layers is also a key element. Thus, the model becomes spatially tolerant: similar representations are learned in different regions of the input, and the total number of parameters can also be reduced drastically.

Deep Learning refers to the increased depth of neural networks. Intuitively, it is expected that neural networks with many hidden layers are more powerful than shallow ones with a single hidden layer. However, as the number of layers increases the training may become surprisingly hard, partly because the gradients are unstable. Batch normalization is a technique to overcome this problem; it reduces internal covariance shift and helps to smooth learning. The main idea of batch normalization is to bring back the benefits of normalization at each layer [32]. Batch normalization results in faster convergence as well. E.g., with batch normalization 7% of the training steps were enough to achieve similar accuracy in an image classification task [32]. Moreover, an additional advantage of batch normalization is that it regularizes the training and thus reduces the need for dropout and other regularization techniques [32]. However, batch normalization and dropout are often simultaneously applied.

Convolutional neural networks were successfully applied to various NLP tasks [19, 30, 31, 52]. These results suggest investigating the possibility of applying CNN based sequence-to-sequence models for G2P. We expected that the advantage of convolutional neural networks enhances the performance of G2P conversion. As known, LSTMs read input sequentially, the output for further inputs depends on the previous ones. Thus, we cannot parallelise these networks. Applying CNN also moves away computational load by using large receptive fields.

Deep neural networks with a sequential architecture have many typical building blocks, such as convolutional or fully connected layers, stacked on each other. Increasing the number of layers in these kinds of networks does not implicitly mean improved accuracy (in our case PER or WER), and some issues, such as vanishing gradient and degradation problems arise as well. Introducing residual and highway connections can improve performance significantly [33, 34]. These connection alternatives allow the information to flow more into the deeper layers, increase the convergence speed and decrease the vanishing gradient problem.

3.1. Models

Encoder-decoder structures have shown state-of-the-art results in different NLP tasks [13, 21]. The main idea of these approaches has two steps: the first step is mapping the input sequence to a vector; the second step is to generate the output sequence based on the learned vector representation. Encoder-decoder models generate an output after the complete input sequence is processed by the encoder, which enables the decoder to learn from any part of the input without being limited to fixed context windows. Fig. 1 shows an example of an encoder-decoder architecture [12].

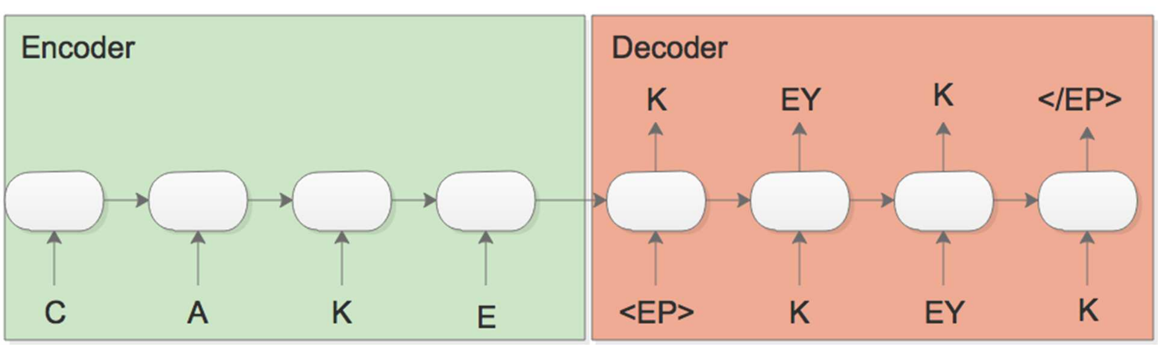


Figure 1. The input of the encoder is "**CAKE**" grapheme sequence, and the decoder produces "**K EY K**" as phoneme sequences. The left side is encoder; the right side is decoder. The model stops making predictions after generating the end-of-phonemes tag. As distinct from [12, 13], input data for the encoder is not reversed in all our models.

In our experiments, we used encoder-decoder architectures. Several models with different hyperparameters were developed and tested. From a large number of experiments, five models with the highest accuracy and diverse architectures were selected. Our first two models are based on existing solutions for comparison purposes. We used these models as a baseline. In the following paragraphs the five models are introduced:

1. The first model uses LSTMs for both the encoder and the decoder. The LSTM encoder reads the input sequence and creates a fixed-dimensional vector representation. The second LSTM is the decoder, and it generates the output. Fig. 2(a) shows the structure of the first model. It can be seen that both LSTMs have 1024 units; softmax activation function is used to obtain model predictions. This architecture is the same as a previous solution [13], while the parameters of training (optimization method, regularization, etc.) are identical to the settings used in case of the other four models. This way we try to ensure a fair comparison among the models.

Although the encoder-decoder architecture achieves competitive results on a wide range of problems, it suffers from the constraint that all input sequences are forced to be encoded to a fixed size latent space. To overcome this limitation, we investigated the effects of the attention mechanism proposed by [49, 50] in Model 1 and Model 2. We applied an attention layer between the encoder and decoder LSTMs in case of Model 1, and Bi-LSTMs for Model 2. The introduced attention layers are based on global attention [50].

2. In the second model, both the encoder and the decoder are Bi-LSTMs [10, 35, 36]. The structure of this model is presented in Fig. 2(b). The input is fed to the first Bi-LSTM (encoder), which combines two unidirectional LSTM layers that process the input from left-to-right and right-to-left. The output of the encoder is given as input for the second Bi-LSTM (decoder). Finally, the softmax function is applied to generate the output of one-hot vectors (phonemes). During the inference, the complete input sequence is processed by the encoder, and after that, the decoder generates the output. For predicting a phoneme, both the left and the right contexts are considered. This model was also inspired by an existing solution [11].

3. In the third model, a convolutional neural network is introduced as encoder, and a Bi-LSTM as decoder. This architecture is presented in Fig. 2(c). As this figure shows the number of filters is 524, the length of the filter is 23, the stride is 1, and the number of cells in the Bi-LSTM is 1024. In this model, the CNN layer takes graphemes as input and performs convolution operations. For regularization purpose, we also introduced batch normalization in this model.

4. The fourth model contains convolutional layers only with residual connections (blocks) [33]. These residual connections have two rules [37]:

(1) if feature maps have the same size, then the blocks share the same hyperparameters.

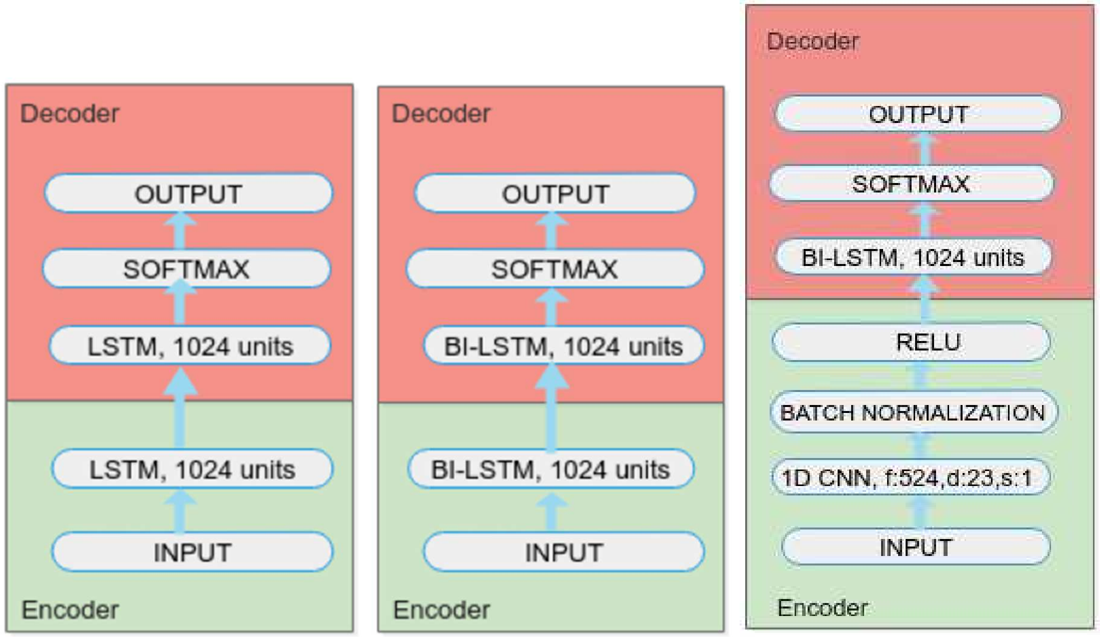
(2) each time when the feature map is halved, the number of filters is doubled.

First, we apply one convolutional layer with 64 filters to the input layer, followed by a stack of residual blocks. Through hyperparameter optimization, the best result was achieved by 4 residual blocks, as shown in Fig. 3(a) and the number of filters in each residual block is 64, 128, 256, 512, respectively. Each residual block contains a sequence of two convolutional layers followed by a batch normalization [32] layer and ReLU activation. The filter size of all convolutional layers is three. After these blocks, one more batch normalization layer and ReLU activation are applied. The architecture ends with a fully connected layer, which uses the softmax activation function.

We carried out experiments with the same fully convolutional models without residual connections, however, the phoneme and word error rates were worse than with residual connections, as expected.

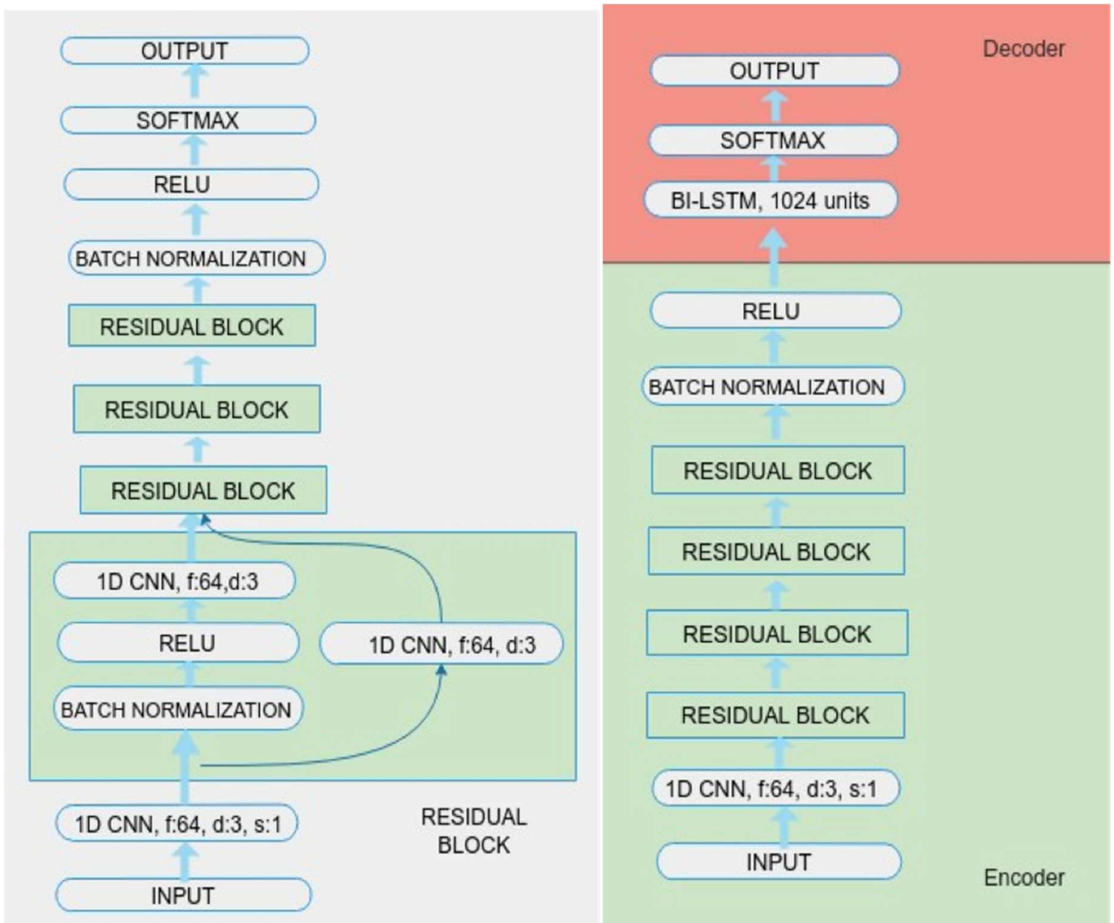
5. The fifth model combines models 3 and 4: the encoder has the same convolutional neural network architecture with residual connections and batch normalization, which was introduced in model four. The decoder is a Bi-LSTM, as in model three. The structure of this model is presented in Fig. 3(b).

In all models except Model 4, we used stateless LSTM (or Bi-LSTM) configurations; the internal state is reset after each batch for predictions.



a) b) c)

Figure 2. G2P conversion model based on encoder-decoder (a) LSTMs (first model); (b) Bi-LSTMs (second model); (c) encoder CNN, decoder Bi-LSTM (third model). **f, d, s** are the number of the filters, length of the filters and stride, respectively, in the convolutional layer.



a) b)

Figure 3. G2P conversion based on (a) convolutional neural network with residual connections (fourth model) and (b) encoder convolutional neural network with residual connections and decoder Bi-LSTM (fifth model). **f, d, s** is the number of the filters, length of the filters and stride, respectively.

3.2. Details of the bidirectional decoder

The details of the bidirectional decoder, which was used in Model 2, are presented in this section.

Given an input sequence $x = (x_1, x_2, .., x_N)$, LSTM network computes the hidden vector sequence $h = (h_1, h_2, .., h_N)$ and output vector sequence $y = (y_1, y_2, .., y_N)$.

Initially, one-hot character vectors for graphemes and phonemes sequences were created. Character vocabularies, which contain all the elements that are present in the input and output data, are separately calculated. In other words, neither any grapheme vector in the output vocabulary, nor any phoneme vector in the input vocabulary was used. These were the inputs to the encoder and the decoder. Padding was applied to make all input and output sequences to have the same length, which was set to 22. This number (22) was chosen based on the maximum length in the training database. For G2P, $x = (x_1, x_2, .., x_N)$ is one-hot character vectors of graphemes sequences; $y = (y_1, y_2, .., y_N)$ is one-hot character vectors of phonemes sequences.

In the proposed Model 2, as an encoder, Bi-LSTM was used, and it consists of two LSTMs: one that processes the sequence from left-to-right (forward encoder), and one that does it in reverse (backward encoder). It was applied to learn the semantic representation of the input sequences in both directions. One LSTM looks at the sequence from left-to-right (forward encoder), so reads an input sequence in left-to-right order; and another LSTM looks at it in reverse (backward encoder), so reads an input sequence in a right-to-left order. Each of the time steps the forward hidden sequence $\vec{h}$ and the backward hidden sequence $\overleftarrow{h}$ are iterated by the following equations [48]:

$$\overrightarrow{h_t} = \mathcal{H}(W_{x\vec{h}}x_t + W_{\vec{h}\vec{h}}\vec{h}_{t-1} + b_{\vec{h}}) \quad (1)$$

$$\overleftarrow{h_t} = \mathcal{H}\left(W_{x\overleftarrow{h}}x_t + W_{\overleftarrow{h}\overleftarrow{h}}\overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right) \quad (2)$$

In Equation (1) the forward layer iterated from $t = 1$ to N; in Equation (2) the backward layer is iterated from $t = N$ to 1; $\mathcal{H}$ is an element-wise sigmoid function.

As next step, the hidden states of these two LSTMs were concatenated to form an annotation sequence $h = \{h_1, h_2, .., h_N\}$, where $h_t = [\overrightarrow{h_t}, \overleftarrow{h_t}]$ encodes information about the $t-$th grapheme with respect to all the other surrounding graphemes in the input. $W_{x\vec{h}}$, $W_{x\overleftarrow{h}}$, $W_{\overleftarrow{h}\overleftarrow{h}}$ and $W_{\vec{h}\vec{h}}$ are weight matrixes; $b_{\vec{h}}, b_{\overleftarrow{h}}$ denotes the bias vectors. Generally, in all parameters, the arrow which pointed left to right and right to left means forward and backward layer, respectively.

The forward LSTM unrolls the sequences until it reaches the end of sequence for that input. The backward LSTM unrolls the sequences until it reaches the start of the sequence.

For the decoder, we used bidirectional LSTM. These LSTMs can be called forward and backward decoder, and described as $\vec{d}, \overleftarrow{d}$. After concatenating the forward and backward encoder LSTMs, the backward decoder performs decoding in a right-to-left way. It was initialized with final encoded state and reversed output (phonemes). The forward decoder is trained to sequentially predict the next phoneme given the phoneme sequence. This part was initialized with the final state of the encoder and all phoneme sequences.

Each decoder output is passed through the softmax layer that will learn to classify the correct phonemes.

For training, given the previous phonemes, the model factorizes the conditional into a summation of individual log conditional probabilities from both directions,

$$P\left(y_t \middle| y_{[1:(t-1)]}, y_{[(t+1):N]}\right) = \overrightarrow{\log P(y_t|y_{1:t-1})} + \overleftarrow{\log P\left(y_t \middle| y_{(t+1):N}\right)} \quad (3)$$

Where $\overrightarrow{\log P\left(y_t \middle| y_{[1:(t-1)]}\right)}$ and $\overleftarrow{\log P\left(y_t \middle| y_{[(t+1):N]}\right)}$ are the left-to-right (forward), the right-to-left (backward) conditional probability in Equation (3), and calculated as below equations:

$$\overleftarrow{\log P\left(y_t \middle| y_{[(t+1):N]}\right)} = \sum \log P\left(y_t \middle| \{y_{t+1}, .., y_N\}, x, h, \overleftarrow{d}\right) \quad (4)$$

$$\overrightarrow{\log P\left(y_t \middle| y_{[1:(t-1)]}\right)} = \sum \log P\left(y_t \middle| \{y_1, .., y_{t-1}\}, x, h, \vec{d}\right) \quad (5)$$

The prediction is performed on test data as follows:

$$\overleftarrow{\log P\left(y_t \middle| y_{[(t+1):N]}\right)} = \sum \log P(y_t|x, h]) \quad (6).$$

According to Equation (6) future output is not used during inference. The architecture is shown on Figure 4.

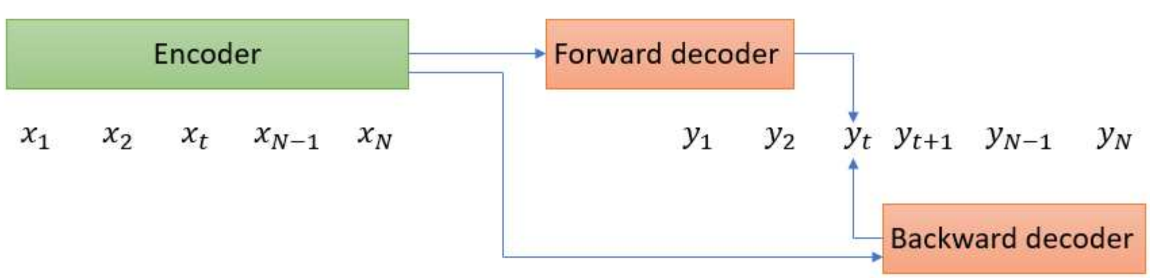


Figure 4. The architecture of the proposed bidirectional decoder model for G2P task.

4. Experiments

4.1. Datasets

We used the CMU pronunciation[4] and NetTalk[5] datasets, which have been frequently chosen by various researchers [3, 13, 23]. The training and testing splits are the same as found in [4, 5, 8, 11], thus, the results are comparable. CMUDict contains a 106,837-word training set and a 12,000-word test set (reference data). 2,670 words are used as development set. There are 27 graphemes (uppercase alphabet symbols plus the apostrophe) and 41 phonemes (AA, AE, AH, AO, AW, AY, B, CH, D, DH, EH, ER, EY, F, G, HH, IH, IY, JH, K, L, M, N, NG, OW, OY, P, R, S, SH, T, TH, UH, UW, V, W, Y, Z, ZH, <EP>, </EP>) in this dataset. NetTalk contains 14,851 words for training, 4,951 words for testing and does not have a predefined validation set. There are 26 graphemes (lowercase alphabet symbols) and 52 phonemes ('!', '#', '*', '+', '@', 'A', 'C', 'D', 'E', 'G', 'I', 'J', 'K', 'L', 'M', 'N', 'O', 'R', 'S', 'T', 'U', 'W', 'X', 'Y', 'Z', '^', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'k', 'l', 'm', 'n', 'o', 'p', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z',<EP>, </EP>) in this dataset.

We use **<EP>** and **</EP>** tokens as beginning-of-graphemes and end-of-graphemes tokens in both datasets. For inference, the decoder uses the past phoneme sequence to predict the next phoneme, and it stops predicting after token **</EP>**.

4.2. Training

For the CMUDict experiments, in all models, the size of the input layers is equal to

input: {length of the longest input (22) X number of graphemes (27)}

and the size of the output layers is equal to the

output: {length of the longest output (22) X number of phonemes (41)}

In order to transform graphemes and phonemes for neural networks, we convert inputs to 27-dimensional and outputs to 41-dimensional one-hot vector representations. For example, the phoneme sequences of the word 'ARREST' is 'ER EH S T'; the input and output vector of the grapheme and phoneme sequences are as below:

Input vector of ‘ARREST’ :

$$\begin{array}{cc} & \begin{array}{cccccccccc} A & B & C & E & \cdots & R & \cdots & S & T & \cdots \end{array} \\ \begin{array}{c} A \\ R \\ R \\ E \\ S \\ T \\ \ldots \end{array} & \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & \ldots & 0 \end{bmatrix} \end{array}$$

Output vector of 'ER EH S T':

$$\begin{array}{cc} & \begin{array}{cccccccccc} /EP & \cdots & EH & ER & EP & \cdots & S & SH & T & \cdots \end{array} \\ \begin{array}{c} EP \\ ER \\ EH \\ S \\ T \\ /EP \\ \ldots \end{array} & \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \ldots & 0 & 0 & 0 & \ldots & \ldots & 0 \end{bmatrix} \end{array}$$

In case of LSTMs we applied the Adam optimization algorithm [39] with a starting learning rate of 0.001, and with the baseline values of β_1, β_2 and ε (0.9, 0.999 and $1e^{-08}$, respectively). For batch size 128 was chosen. Weights were saved when the PER on the validation dataset achieved a lower value

---

4 http://www.speech.cs.cmu.edu/cgi-bin/cmudict

5 We are grateful to Stan Chen for providing the data

than before. When the PER did not decrease further within 100 epochs, the best model was chosen, and it was trained with stochastic gradient descent (SGD) further. In case of the first, second and third models for SGD we used 0.005 for learning rate, 0.8 for momentum. For the fourth (convolutional with residual connections) model 0.05 (learning rate) and 0.8 (momentum) were applied, and it was trained for 142 when early stopping was called. In the fifth model 0.5 (learning rate) of SGD and 0.8 (momentum) was set and when PER has stopped improving in about 50 epochs, the learning rate was multiplied by 4/5. The number of epochs for this model reached 147 and 135 for CMUDict and NetTalk, respectively.

In all proposed models, the patience of early stopping was set to 50 in Adam optimizer and 30 in SGD optimizer.

For NetTalk experiments, and the size of input and output layers are equal to

input: {length of the longest input (19) X number of graphemes (26)}

output: {length of the longest output (19) Xnumber of phonemes (52)}.

We converted inputs to 26-dimensional and outputs to 52-dimensional one-hot vector representations as in case of CMUDict. The same model structure was used as with the CMUDict experiments.

Moreover, the implementation of a single convolutional layer on input data is presented in Fig. 5. The input is a one-hot vector of 'ARREST'; 64 filters of (input length) *3 are applied to the input. In other words, the input is convolved with 64 feature maps, which produce the output of the convolutional layer. Zero padding was used to ensure that the output of the convolution layer has the same dimension as the input. During training, the filter weights are optimized to produce lower loss values.

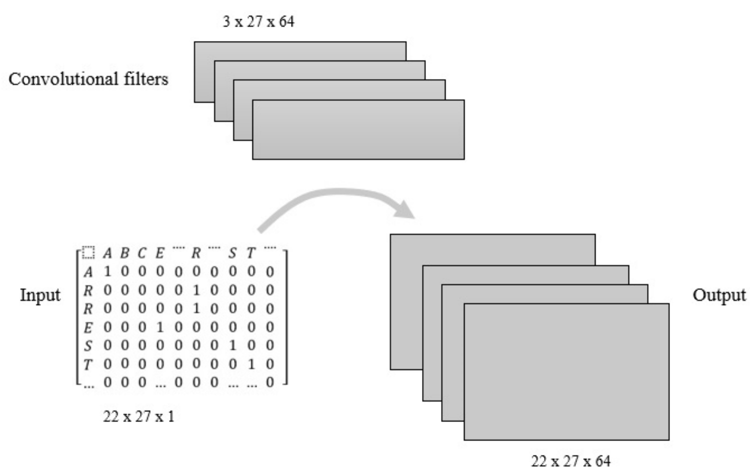


Figure 5. Implementation of a single convolutional layer with 64 filters of size (input length) *3 to the input data.

During inference, prediction of the graphemes sequence is decoded until **/EP**, and the length of input and output are not considered.

4.3. Evaluation and Results

NVidia Titan Xp (12 GB) and NVidia Titan X (12 GB) GPU cards hosted in two i7 workstations with 32GB RAM served for training and inference. Ubuntu 14.04 with Cuda 8.0 and cuDNN 5.0 was used as general software architecture. For training and evaluating the Keras deep learning framework with Theano backend was our environment.

For evaluation standard and commonly used [11, 13] measurements of phoneme error rate (PER) and word error rate (WER) were calculated. PER was used to measure the distance between the predicted phoneme sequence and reference pronunciation divided by the number of phonemes in the reference pronunciation. Edit distance (also known as Levenshtein distance [38]) is the minimum number of insertions (I), deletions (D) and substitutions (S), that are required to transform one sequence into the other. If there are multiple pronunciation variants for a word in the reference data, the variant that has the smallest Levenshtein distance [38] to the candidate is used. Levenshtein distance can be calculated by dynamic programming method [47].

For WER computation, which is only counted if the predicted pronunciation does not match any reference pronunciation, the number of word errors is divided by the total number of unique words in the reference.

After training the model, predictions were run on the test dataset. The results of evaluation on the CMUDict dataset are shown in Table 1. The first and second columns show the model number and the applied architecture, respectively. The third and fourth columns show the PER and WER values. The fifth column of Table 1 contains the average sum of training and validation time of one epoch. The last two columns present information about the size of models, which shows the number of parameters (weights) and the number of epochs to reach minimum validation loss. According to the results, the encoder-decoder Bi-LSTM architecture (Model 2) outperforms the first model, as expected. But attention-based Model 1 (called Model 1A in Table 1) outperforms Model 2 in term of PER. The best WER and PER values are achieved by the fifth model: PER is 4.81%, and WER is 25.13%. Attention-based Model 2 (called Model 2A in Table 1) approaches the best result in terms of both PER and WER. But the number of parameters of Model 2A is twice as much as for Model 5. Although the fourth model was faster than all other models, both PER and WER of this model are the highest, however, still competitive. Moreover, this model has also the least parameters.

We compared the performance of the fifth model on both CMUDict and NetTalk with previously achieved state-of-the-art results. These comparisons are presented in Table 2. The first column shows the dataset, the second column presents the method used in previous solutions with references, PER and WER columns tell the results of the referred models. Table 2 clearly shows that our fifth model outperforms the previous solutions by PER on each dataset, except [23]. For NetTalk, we are able to significantly surpass the previous state-of-the-art, but a better WER was obtained by [23] with an encoder-decoder network based on attention mechanism. We should point out that the results of the fifth model are very close to those obtained by [23].

The proposed best model in [40] consists of the combination of the sequitur G2P (model order 8) and seq2seq-attention (Bi-LSTM 512x3) and multitask learning (ARPAbet/IPA), and although the WER in their case is better, Model 5 has the smaller PER.

Although the encoder-decoder LSTM by [13] is similar to our first model, the PER is better in our case; the WER of both models is almost the same. Our second model is comparable with [13], in which the Bi-LSTM method was implemented, alignment was also applied.

Table 1. Results on the CMUDict dataset.

Model	Method	PER(%)	WER(%)	Time(s)	Number of epochs	Model size
1	Encoder-Decoder LSTM	5.68	28.44	467.73	185	12.7M
1A	Encoder-Decoder LSTM with attention layer	5.23	28.36	688.9	136	13.9M
2	Encoder-Decoder Bi-LSTM	5.26	27.07	858.93	177	33.8M
2A	Encoder-Decoder Bi-LSTM with attention layer	4.86	25.67	1045.5	114	35.3M
3	Encoder CNN, decoder Bi-LSTM	5.17	26.82	518.3	115	13.1M

4	End-to-end CNN (with res. connections)	5.84	29.74	176.1	142	7.62M
5	Encoder CNN with res. connections, decoder Bi-LSTM	4.81	25.13	573.5	147	14.5M

Table 2. Comparison of best previous results of G2P models with our fifth model (encoder is a CNN with residual connections, Bi-LSTM decoder) on CMUDict and NetTalk.

Data	Method	PER (%)	WER (%)
NetTalk	Joint sequence model [3]	8.26	33.67
	Bi-LSTM [13]	7.38	30.77
	Encoder-decoder with global attention [23]	7.14	29.20
	Encoder CNN with residual connections, decoder Bi-LSTM (Model 5)	**5.69**	**30.10**
CMUDict	LSTM with Full-delay [11]	9.11	30.1
	Joint sequence model [3]	5.88	24.53
	Encoder-decoder LSTM [13]	7.63	28.61
	Bi-LSTM +Alignment [13]	5.45	23.55
	Combination of sequitur G2P and seq2seq-attention and multitask learning [40]	5.76	24.88
	Ensemble of 5 [Encoder-decoder + global attention] models [23]	4.69	20.24
	Encoder-decoder with global attention [23]	5.04	21.69
	Joint multi-gram + CRF [8]	5.5	23.4
	Joint n-gram model [4]	7.0	28.5
	Joint maximum entropy (ME) n-gram model [5]	5.9	24.7
	Encoder-Decoder GRU [15]	5.8	28.7
	Encoder CNN with residual con., decoder Bi-LSTM (fifth model)	**4.81**	**25.13**

5. Discussions

In this section, we discuss the results of the previous section and analyse the connection between PER values and word length, furthermore the position of the error within the word.

We categorize the word length into 3 classes: short (shorter than 6 characters), medium (between 6 and 10 characters), long (more than 10 characters). According to this categorization, there were 4306 short, 5993 medium and 1028 long words in the CMUDict dataset. In this analysis, we ignored approximately 600 words that have multiple pronunciation variants in the reference data.

The result of this comparison is presented in Fig. 6(a). For short words, all models show similar PERs; for medium length words, except the end-to-end CNN model (fourth model), the other models resulted in similar error; for long words, encoder CNN with residual connection, decoder Bi-LSTM (fourth model) and encoder CNN, decoder Bi-LSTM (third model) got similar minimum errors. The fourth model showed the highest error in both medium and long length words. According to Fig. 6(a), the advantage of Bi-LSTM based models is clearly shown for learning long sequences.

Moreover, errors occurring in the first half of the pronunciation (in the reference) increases the probability of predicting incorrect phonemes in the second half. Still, a correctly predicted first half cannot guarantee a correctly predicted second half. In our experiments, convolutional architectures also performed well on short and on long-range dependencies. Our intuition is that the residual connections enable the network to consider features learned by lower and higher layers - which represents shorter and longer dependencies.

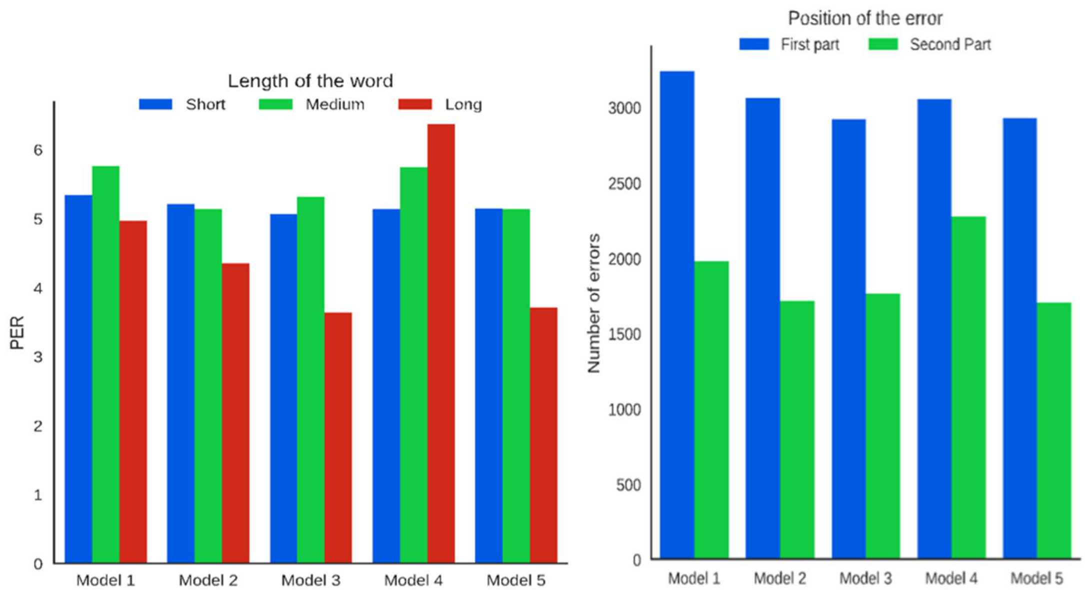
We also analysed the position of the errors in the reference pronunciation: we investigated if the error occurred in the first or in the second half of the word. The type of error can be insertion (I), deletion (D) and substitution (S). By using this position information, we can analyse the distribution of these errors across the first or second half of the word. The position of error was calculated by enumerating graphemes in the reference. For error insertion (I), the position of the previous grapheme was taken into account. The example below describes the process details:

word: ACKNOWLEDGEMENT
Enumeration: 0 1 2 3 4 5 6 7 8 9 10 11 12
Reference: [EP AE K N AA L IH JH M AH N T /EP]
Prediction: [EP IH K N AA L IH JH IH JH AH N T /EP]
Type of errors: S S I
Position: [1, 8, 8]

As the example shows two substitutions (S) and one insertion (I) occurred in our fifth model output. One error (S) is included in the first half part of the pronunciation in the reference (EP AE K N AA L, the other errors (S) and (I) are in the second half (H JH M AH N T /EP).

Fig. 6(b) shows the position errors calculated for all the models on the reference dataset. The first half of the words in all models contains more errors. Regarding the second half, all models show a similar number of position errors, except the end-to-end CNN model. The fifth model resulted in the lowest number of position errors.

Furthermore, in all models presented here, PER is better than the previous results on CMUDict except the first four models in [23] while WER is still reasonable. It means that even most of the incorrect predictions are very close to the reference, therefore they have small PER. Accordingly, we need to analyse the incorrect predictions (outputs) for each model to see how many phonemes are correct in the reference. In the fifth model, 25.3 % of the test data are not correct (about 3000 test samples). After the analysis of these predictions, more than half of them have 1 incorrect phoneme. In particular, the PER for 59 test samples is higher than 50% (11 test samples are greater than 60%, and only 1 test sample is more than 70%). These percentages in the other presented models are more or less the same. Generally, the same 1000 words are incorrectly predicted by all presented models.



a) b)

Figure 6. PER depending on the length of the words (a); Position of errors for all models (b).

We can see different types of error when generating phoneme sequences. One of these errors is that some phonemes are unnecessarily generated multiple times. For example, for the word YELLOWKNIFE, reference is [Y EH L OWN AY F], the prediction of Model 5 for this word is [Y EH L OW K N N F], where the character N was generated twice. Another error type regards sequences of graphemes that are rarely represented in the training process. For example, for the word ZANGHI Model 5 output is [Z AE N G], while the reference is [Z AA N G IY]. The graphemes 'NGHI' appeared only 7 times in the training data. Furthermore, many words are of foreign origin, for example, GDANSK is Polish a city, SCICCHITANO is an Italian name, KOVACIK is a Turkish surname. Generating phoneme sequences of abbreviations is one of the hard challenges. For example, LPN, INES are shown with their references and the prediction form of Model 5 in Table 3:

Table 3. Examples of errors predicted by Model 5.

Word from test data	Reference of given word	Prediction of Model 5
YELLOWKNIFE	Y EH L OW N AY F	Y EH L OW K N N F
ZANGHI	Z AA N G IY	Z AE N G
GDANSK	G AH D AE N S K	D AE N AE K EH K
SCICCHITANO	S IH K AH T AA N OW	S CH CH Y K IY IY
KOVACIK	K AA V AH CH IH K	K AH V AA CH IH K
LPN	EH L P IY EH N	L L N N P IY E
INES	IH N IH S	AY N Z

In the proposed models, we were able to achieve smaller PERs with different hyperparameter settings, but WERs showed different behaviour, in contrast, what we expected. For calculating WER, the number of word errors is divided by the total number of unique words in the reference. These word errors are counted only if the predicted pronunciation does not match any reference pronunciation. So, in the generated phoneme sequences of words that contained errors, there is at least one phoneme error. For that reason, we calculated the number of word errors depending on the number of phoneme errors for all proposed models on CMUDict, as presented in Fig. 7.

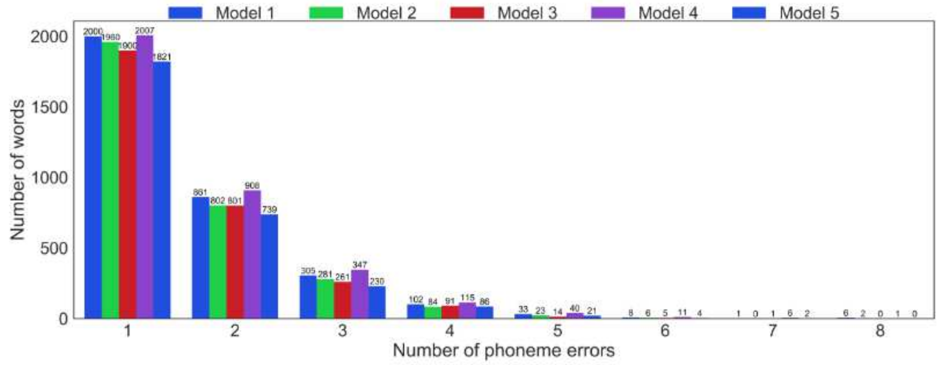


Figure 7. Number of word errors depending on the number of phoneme errors for all models.

In the case of each model, there are twice as many words with only one phoneme error than words which have two phoneme errors. Words with one phoneme error significantly effect the WER. The number of words with two phoneme errors were the most in Model 4 (908), and the least in Model 5 (739). The number of words, which have three phoneme errors is the least (230) in Model 5. There is approximately the same number of words which have four phoneme errors in Model 2 and Model 5 (84 in Model 2 and 86 in Model 5). There are very few words with five or more phoneme errors in all models. Model 1, Model 3 has only 1 word which has seven phoneme errors; Model 5 has 2 words; Model 4 has 6 words. The number of words with eight phoneme errors is 0 in Model 3, Model 5; 1 in Model 4. Fig. 7 helps to understand why PER in our models can be smaller while WER is higher.

6. Conclusions

In this paper convolutional neural networks for grapheme-to-phoneme conversion are introduced. Five different models for the G2P task are described, and the results are compared to previously reported state-of-the-art research. Our models are based on the seq2seq architecture, and, in the fourth and fifth models, we applied CNNs with residual connections. The fifth model, which uses convolutional layers with residual connections as encoder and Bi-LSTM as decoder outperformed most the previous solutions on the CMUDict and NetTalk datasets in terms of PER. Furthermore, the fourth model, which contains convolutional layers only, is significantly faster than other models and still has competitive accuracy. Our solution achieved these results without explicit alignments. The experiments are conducted on a test set, which is 9.8% and 24.9% of the whole CMUDict and NetTalk databases, respectively. The same test set is used in all cases, so we consider the results comparable. To draw conclusions on whether one model is better than another the goal must be defined. If inference time is crucial, then smaller model sizes are favorable (e.g. Model 4), but if lower WER and PER are the main factors, then Model 5 outperforms the others.

The results presented in this paper can be applied in TTS systems, however, because of the rapid development of deep learning further aspects will be investigated, like dilated convolutional networks and neural architecture search. These are possible further extensions of the current research.

Abbreviations

G2P: Grapheme-to-phoneme

ASR: Automatic Speech Recognition

CNN: Convolutional neural network

PER: Phoneme error rate
WER: Word error rate

Bi-LSTM: bi-directional Long Short Term Memory

Declarations

Acknowledgements

The research presented in this paper has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013), by the BME-Artificial Intelligence FIKP grant of Ministry of Human Resources (BME FIKP-MI/SC), by Doctoral Research Scholarship of Ministry of Human Resources (ÚNKP-18-4-BME-394) in the scope of New National Excellence Program, by János Bolyai Research Scholarship of the Hungarian Academy of Sciences, by the VUK project (AAL 2014-183), and the DANSPLAT project (Eureka 9944). We gratefully acknowledge the support of NVIDIA Corporation with the donation of the Titan Xp GPU used for this research.

Authors' contributions

All authors have read and approved the final manuscript.

Ethics approval and consent to participate

Not applicable.

Competing interests

The authors declare that they have no competing interests.

Authors' Affiliations

(1, 2, 3) Department of Telecommunications and Media Informatics, University of Budapest Technology and Economics.

References

1. Elovitz, H., Rodney, J., McHugh, A., & Shore, J. (1976). Letter-to-Sound Rules for Automatic Translation of English Text to Phonetics. IEEE Transactions on Acoustics, Speech, and Signal Processing, 24 (6): 446–459, doi:10.1109/TASSP.1976.1162873.
2. Black, A. W., Lenzo, K., & Page, V. (1998). Issues in Building General Letter to Sound Rules. Proceedings of the 3rd ESCA Workshop on Speech Synthesis, 77–80.
3. Bisani, M., & Ney, H. (2008). Joint-Sequence Models for Grapheme-to- Phoneme Conversion. Speech

Communication, 50 (5): 434–451, doi: 10.1016/j.specom.2008.01.002.
4. Galescu, L., & F. Allen, J. (2002). Pronunciation of Proper Names with a Joint N-Gram Model for Bi-Directional Grapheme-to-Phoneme Conversion. 7th International Conference on Spoken Language Processing, 109–112.
5. Chen, S. F. (2003). Conditional and Joint Models for Grapheme-to-Phoneme Conversion. 8th European Conference on Speech Communication and Technology, (5): 2033–2036.
6. Lehnen, P., Allauzen, A., Laverge, T., Yvon, F., Hahn, S., & Ney, H. (2013). Structure Learning in Hidden Conditional Random Fields for Grapheme-to-Phoneme Conversion. Proceedings of the Annual Conference of the International Speech Communication Association, 2326–2330.
7. Lehnen, P., Hahn, S., Guta, V., & Ney, H. (2012). Hidden Conditional Random Fields with M-to-N Alignments for Grapheme-to-Phoneme Conversion. Proceedings of the 13th Annual Conference of the International Speech Communication Association, 2554-2557.
8. Wu, K., Allauzen, C., Hall, K., Riley, M., & Roark, B. (2014). Encoding Linear Models as Weighted Finite-State Transducers. Proceedings of the Annual Conference of the International Speech Communication Association, 1258–1262.
9. Yao, K., Peng, B., Zhang, Y., Yu, D., Zweig, G., & Shi, Y. (2014). Spoken Language Understanding Using Long Short-Term Memory Neural Networks. Proceedings of the Spoken Language Technology Workshop (SLT'14), 189–194, doi:10.1109/slt.2014.7078572.
10. Schuster, M., & Paliwal, K., K. (1997). Bidirectional Recurrent Neural Networks. IEEE Transactions on Signal Processing, 45 (11): 2673–2681, doi:10.1109/78.650093.
11. Rao, K., Peng, Fuchun, Sak, H., & Beaufays F. (2015). Grapheme-to-Phoneme Conversion Using Long Short-Term Memory Recurrent Neural Networks. IEEE International Conference on Acoustics, Speech and Signal Processing, 4225–4229.
12. Sutskever, I., Vinyals, O., & V. Le, Q. (2014). Sequence to Sequence Learning with Neural Networks. Advances in Neural Information Processing Systems (NIPS), 3104–3112.
13. Yao, K., & Zweig, G. (2015). Sequence-to-Sequence Neural Net Models for Grapheme-to-Phoneme Conversion. Proceedings of the Annual Conference of the International Speech Communication Association, 3330–3334.
14. Mousa, A., E., &, Schuller, B. (2016). Deep Bidirectional Long Short-Term Memory Recurrent Neural Networks for Grapheme-to-Phoneme Conversion Utilizing Complex Many-to-Many Alignments. Proceedings of the Annual Conference of the International Speech Communication Association, 2836–2840, doi:10.21437/Interspeech.2016-1229.
15. Arik, S.Ö., Chrzanowski, M., Coates, A., Diamos, G., Gibiansky, A., Kang, Y., Li, X., Miller, J., Raiman, J., Sengupta, S., & Shoeybi, M (2017). Deep Voice: Real-Time Neural Text-to-Speech. Proceedings of the 34th International Conference on Machine Learning, PMLR 70, 195-204.
16. Krizhevsky, A., Sutskever, I., & E. Hinton, G. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Proceedings of the 25th International Conference on Neural Information Processing Systems (NIPS'12), (1), 1097-1105.
17. Zeiler, M. D. & Fergus, R. (2014). Visualizing and understanding convolutional networks. 13th European Conference on Computer Vision (ECCV), 818-833, doi: 10.1007/978-3-319-10590-1_53
18. Gehring, J., Auli, M., Grangier, D., & Dauphin, Y (2016). A Convolutional Encoder Model for Neural Machine Translation. Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, 123-135, doi:10.18653/v1/p17-1012
19. Gehring, J., Auli, M., Grangier, D., Yarats, D., & Dauphin, Y. (2017). Convolutional Sequence to Sequence Learning. arXiv preprint arXiv: 1705.03122.
20. Kalchbrenner, N., & Blunsom, P. (2013). Recurrent Continuous Translation Models. Proceedings of the 2013

Conference on Empirical Methods in Natural Language Processing, 1700–1709, doi:10.1146/annurev.neuro.26.041002.131047.

21. Cho, K., Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations Using RNN Encoder-Decoder for Statistical Machine Translation. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, 1724-1734, doi:10.3115/v1/d14-1179.
22. Lu, L., Zhang, X., & Renals, S. (2016). On Training the Recurrent Neural Network Encoder-Decoder for Large Vocabulary End-to-End Speech Recognition. IEEE International Conference on Acoustics, Speech and Signal Processing, 5060–5064, doi:10.1109/icassp.2016.7472641
23. Toshniwal, S., & Livescu, K. (2016). Jointly learning to align and convert graphemes to phonemes with neural attention models. IEEE Spoken Language Technology Workshop (SLT), doi:10.1109/SLT.2016.7846248
24. Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). Rethinking the Inception Architecture for Computer Vision. Proceedings IEEE Conference on Computer Vision and Pattern Recognition, 2818-2826, doi:10.1109/cvpr.2016.308.
25. Szegedy, C, Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., & Rabinovich, A. (2015). Going Deeper with Convolutions. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1-9, doi:10.1109/cvpr.2015.7298594.
26. Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., C.Berg, A., & Fei-Fei L. (2015). ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision, 115 (3): 211–252, doi:10.1007/s11263-015-0816-y.
27. Bluche, T., Ney, H., & Kermorvant, C. (2013). Tandem HMM with Convolutional Neural Network for Handwritten Word Recognition. Proceedings of the IEEE International Conference on Acoustics Speech and Signal Processing (ICASSP2013), 2390–2394.
28. LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. Proceedings of the IEEE 86 (11): 2278–2323, doi:10.1109/5.726791
29. Schroff, F., Kalenichenko, D., & Philbin, J. (2015). FaceNet: A Unified Embedding for Face Recognition and Clustering. Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 815–823, doi:10.1109/cvpr.2015.7298682.
30. Conneau, A., Schwenk, H., Barrault, L., & Lecun, Y. (2016). Very Deep Convolutional Networks for Text Classification. KI - Künstliche Intelligenz 26 (4): 357–363.
31. Zhou, P., Qi, Z., Zheng, S., Xu, J., Bao, H., & Xu, B. (2016). Text Classification Improved by Integrating Bidirectional LSTM with Two-Dimensional Max Pooling. Proceedings of the 26th International Conference on Computational Linguistics (COLING): Technical Papers, 3485–3495.
32. Ioffe, S., & Szegedy. C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. Proceedings of the 32 International Conference on Machine Learning, PMLR 37, 448-456.
33. He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778, doi:10.1109/CVPR.2016.90.
34. Greff, K., K.Srivastava, R., & Schmidhuber, J. (2017). Highway and Residual Networks Learn Unrolled Iterative Estimation. arXiv preprinted arXiv:1612.07771.
35. Sundermeyer, M., Alkhouli, T., Wuebker, J., & Ney, H., (2014). Translation Modeling with Bidirectional Recurrent Neural Networks. Proceedings of the Conference on Empirical Methods in Natural Language Processing, 14–25, doi:10.3115/v1/d14-1003.
36. Fan, Y., Qian, Y., Xie, F., & Soong, F.K. (2014) TTS Synthesis with Bidirectional LSTM Based Recurrent Neural Networks. Proceedings of the Annual Conference of the International Speech Communication Association, 1964–1968.

37. Saining, X., Girshick, R., Dollar, P., Tu, Z., & He, K. (2017) Aggregated Residual Transformations for Deep Neural Networks. IEEE Conference on Computer Vision and Pattern Recognition (CVPR).
38. Levenshtein, V. I. (1966). Binary Codes Capable of Correcting Deletions, Insertions, and Reversals. Soviet Physics Doklady, 10 (8): 707–710, doi:citeulike-article-id:311174.
39. Kingma, D. P., and Jimmy, L. B. (2015). Adam: A Method for Stochastic Optimization. International Conference on Learning Representations.
40. Milde, B., Schmidt, C., Köhler, J. (2017) Multitask Sequence-to-Sequence Models for Grapheme-to-Phoneme Conversion. Proc. Interspeech 2017, 2536-2540, doi: 10.21437/Interspeech.2017-1436.
41. Wang, W., Xu, S., Xu, B. (2016) First Step Towards End-to-End Parametric TTS Synthesis: Generating Spectral Parameters with Neural Attention. Proc. Interspeech 2016, 2243-2247.
42. Wang, Y., Skerry-Ryan, R., Stanton, D., Wu, Y., Weiss, R.J., Jaitly, N., Yang, Z., Xiao, Y., Chen, Z., Bengio, S., Le, Q., Agiomyrgiannakis, Y., Clark, R., Saurous, R.A. (2017) Tacotron: Towards End-to-End Speech Synthesis. Proc. Interspeech 2017, 4006-4010, doi: 10.21437/Interspeech.2017-1452.
43. Nguyen, D.Q., Nguyen, D.Q, Chu, C.X., Thater, S., & Pinkal, M. (2017). Sequence to Sequence Learning for Event Prediction. Proceedings of the 8th International Joint Conference on Natural Language Processing, 37–42.
44. Graves, A., Fernandez, S., Gomez, F. & Schmidhuber, J. (2016) Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In Proceedings of the 23th International Conference on Machine Learning (ICML 2006), 369–376.
45. Hori, T., Watanabe, S., Zhang, Y., & Chan, W. (2017) Advances in Joint CTC-Attention based End-to-End Speech Recognition with a Deep CNN Encoder and RNN-LM. INTERSPEECH, 949-953.
46. Zhang, X., Su, J., Qin, Y., Liu, Y., Ji, R., & Wang, H. (2018). Asynchronous Bidirectional Decoding for Neural Machine Translation. In Proceedings of Association for the Advancement of Artificial Intelligence (AAAI-2018), 5698--5705.
47. Zhu, R. & Huang, Y. 2017. Efficient privacy-preserving general edit distance and beyond. Cryptology ePrint Archive, Report 2017/683.
48. Graves, A., Jaitly, N., & Mohamed, A. (2013). Hybrid speech recognition with Deep Bidirectional LSTM. 2013 IEEE Workshop on Automatic Speech Recognition and Understanding, 273-278.
49. Luong, T., Pham, H., & Manning, C.D. (2015). Effective Approaches to Attention-based Neural Machine Translation. Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1412–1421.
50. Bahdanau, D., Cho, K., & Bengio, Y. (2015). Neural Machine Translation by Jointly Learning to Align and Translate. CoRR, abs/1409.0473.
51. Prabhavalkar, R., Rao, K., Sainath, T.N., Li, B., Johnson, L., & Jaitly, N. (2017). A Comparison of Sequence-to-Sequence Models for Speech Recognition. Interspeech 2017, 939-943.
52. Bai, S., Kolter, J. Z., & Koltun, V. (2018). Convolutional sequence modeling revisited. ICLR Workshop Track. 1-20.