English-Spanish Translator using Sequence-to-Sequence Models

Sergio López sergiol@kth.se

Mónica Villanueva monicavi@kth.se

Diego Yus diegoyl@kth.se

Abstract

This project uses a Deep Recurrent Neural Network to implement a sequence to sequence model for language in order to translate from English to Spanish. The baseline recurrent network was obtained from Google's TensorFlow library official resources, and modified to work with the WMT dataset that contains English to Spanish translations from the EU parliament. Several potential improvements over the original network are tested and implemented into the network's architecture, in order to find the best performing network according to two measurements, the perplexity of the network and the BLEU score.

1 Introduction

1.1 Language Modeling

In recent years Deep Neural Networks (DNNs) have become increasingly popular in a lot of cutting edge research, including speech recognition [3] or synthesis [13]. However, DNNs have the special characteristic of requiring a feature vector of fixed size as input data, a precondition difficult to meet in some problems involving sequences of data of a-priori unknown length.

One of these fields, language modeling, is the subject of study of this work. Language modeling can be defined as the problem of finding a probability distribution over a sequence of words. The point of interest is proving that Recurrent Neural Networks (RNNs) can model inputs taking into account its context and can be trained to translate complete sentences from an origin to a target language. In this specific case, from English to Spanish.

RNNs are a special type of neural networks. They differ from normal neural networks in that they use information recurrently, so that the state of the network at a time step becomes the input for the next time step. Given a sequence of inputs $(x_1, \ldots, x_T)$, a standard RNN computes a sequence of outputs $(y_1, \ldots, y_T)$ by iterating the following equation:

$$\begin{aligned} h_t &= \text{sigm}(W^{xh} x_t + W^{hh} h_{t-1}) \\ y_t &= W^{yh} h_t \end{aligned} \tag{1}$$

This allows for some persistence in the information, which is the natural way of thinking when we treat sequential information. The main problem with RNN are long-term dependencies. The further two elements with a dependency are in a sequence (like two words in a sentence), the weaker this dependency will be learnt in the parameters of the net, and after some point RNNs are not capable of learning them.

Long Short-Term Memories (LSTMs) are a special type of RNNs that are able to learn these long-term dependencies. They use a cell state, where information can flow along the cell practically unchanged, and some gates that control which of this information is removed or added [6].

30th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

The method we are trying to replicate is the one published by Sutskever et al. [9], where they perform sequence to sequence translation using LSTM deep networks as opposed to phrase-based Statistical Machine Translation (SMT) systems, the previous standard method used for machine translation. This method improves the memory usage and the performance of the translation under certain conditions [2].

However, there are new improvements over the work being replicated in this project. If the reader is interested in the topic more information can be found in Bahdanau et al. [1] that learns to align and translate at the same time to avoid performance deterioration in long sentences and Wu et al. [12] that explains the new model by Google addressing problems like rare words or missing parts of the input.

1.2 Metrics

Several metrics can be used in order to assert the quality of the system and the translation. The ones used in the reference paper are the perplexity [5] of the system, which represents how a probability distribution predicts a sample ($2^{H(X)}$, with $H(X)$ being the entropy), and the BLEU score [7] that measures the correspondence between a machine-generated translation and one performed by a human professional.

1.3 Motivation

It may seem uncommon to focus on a purely text-based project in the context of a speech course, instead of choosing a topic directly related to speech, such as synthesis or speech recognition. However, the extreme importance of systems like the one analyzed for speech related technologies, such as simultaneous translation systems as a subsystem of a greater speech related technology, motivated us to work on the topic.

This also allowed us to work with recurrent neural networks, a hot topic for sequence to sequence models that can also be utilized directly using speech inputs.

2 Method

The first challenge encountered is mapping a sentence of arbitrary length to another sentence of unknown length, that may or may not correspond to the length of the input sentence. For example, the English sentence “I disagree” is formed by 2 words and 10 characters, while its translation in Spanish “No estoy de acuerdo” needs 4 words and 19 characters.

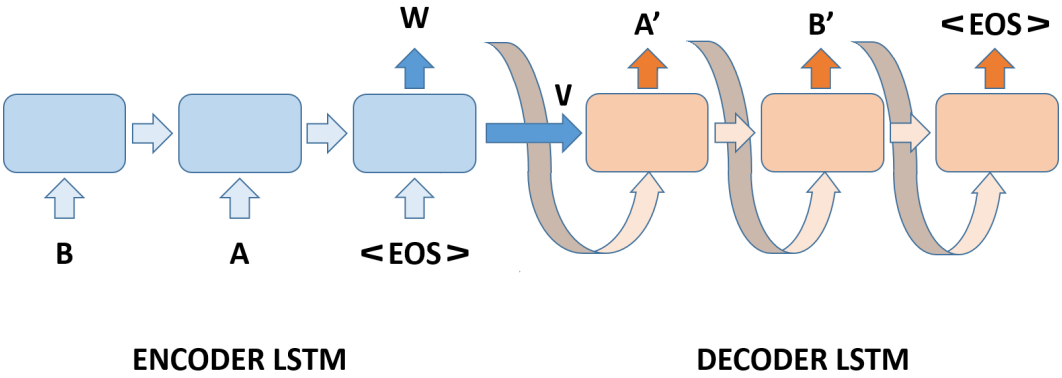

Figure 1: Diagram for the encoding-decoding system using LSTMs. "B A <EOS>" is the inverted input sequence, W is a dummy word generated from the <EOS> (end of sequence), v is the encoded input and "A’ B’ <EOS>" is the target sequence.

The main idea is to encode, one word at a time, an input sentence $(x_1, ..., x_{T_1})$ into a fixed-size vector v of that corresponds to the last hidden state of the first LSTM, the encoder. Then, this vector v

(namely the vector h_{T_1} from Eq. 1) is fed into the second LSTM, the decoder, together with some other input to start the decodification, as depicted in Figure 1. This other input can have two different values: One option is to feed a dummy input "GO" at the first time, and then use the output of the decoder at time t as the input for the decoder at time $t + 1$. The alternative is to directly input the target sentence, even if the decoder made a mistake at the previous time-step.

The second approach is commonly used in training when the correct translations are known so that the decoder can learn better mappings. However, even though it makes the training harder, using the first approach in training can make the system more robust to its own mistakes. The first approach is most frequently used in test, when the sequence is constructed only from the encoded vector [10].

To handle sentences of variable lengths into the encoder, it is necessary a different model for each possible pair length of origin-target sentences. To avoid that, padding is used so that PAD symbols are added at the end of the sentences to balance different lengths. However, if the length difference between short and long sentences is large, a situation could occur in which most of the symbols in a short sentence are paddings. The use of buckets avoid this situation. There are 4 buckets of increasing sizes, where every sentence with length contained in that bucket range will be padded to reach this bucket maximum length. If the sentence does not fit in one bucket, then it will be padded to reach the next bucket maximum size.

Another aspect to be taken in mind is the size of the vocabulary. The vocabulary is the set of (most frequent) unique words considered by our system. Any word outside this vocabulary is tokenized as unknown. Since the computation cost increases greatly with the size of the vocabulary, a compromise must be made to contain all the most frequent words and avoid the ones that are used rarely (like proper names).

As a side note, it is important to mention that the process depicted above takes place on several layers, because we are using a deep LSTM with LSTMs stacked above another. However, the encoder-decoder process is very similar.

The break-through of the method described by Sutskever et al. [9] is, according to them, the inversion of the order of the words of the input sequence. This inversion keeps the average distance between input and output words, however, there is an increased number of short term dependencies because now the first few words in the source sentence are very close to the first few words in the target sentence. Even though it might seem that this inversion could only lead to more confident predictions in the early parts of the target sentence and to less confident predictions in the later parts, the authors reported that LSTMs trained on reversed source sentences did much better on long sentences than LSTMs trained on the raw source sentences.

Mathematically, the idea behind this method is the computation of the conditional probability of the output sequence given the input one, $p(y_1, ..., y_{T_2} \mid x_1, ..., x_{T_1})$. Using LSTMs, this probability can be calculated as follows:

$$p(y_1, ..., y_{T_2} \mid x_1, ..., x_{T_1}) = p(y_{T_2} \mid v, y_1, ..., y_{T_2-1}) \cdot p(y_1, ..., y_{T_2-1} \mid x_1, ..., x_{T_1}) \tag{2}$$

where v is the vector encoding the input sequence, that is, the output of the encoder and T_1, T_2 are the lengths of the input and output sequences respectively. Recursively, the equation can be reduced to

$$p(y_1, ..., y_{T_2} \mid x_1, ..., x_{T_1}) = \prod_{t=1}^{T_2} p(y_t \mid v, y_1, ..., y_{t-1}) \tag{3}$$

The optimization problem solved for learning can be deduced as a logical consequence of Equation 3. If the goal is maximizing the probability of a correct translation, it is possible to define the loss function as the mean of $1 - p(Y \mid X)$ for every pair of sequences in the training set. In order to avoid precision problems, this computation can be moved to log domain using Equation 4

$$L = -\frac{1}{\mid \mathcal{D} \mid} \sum_{Y,X \in \mathcal{D}} \log p(Y \mid X) \tag{4}$$

where $\mathcal{D}$ is the training set.

Once the system is trained, the translation mode only has to return the sequence with highest probability. To avoid a complexity growth, the number of possible hypothesis is pruned using beam search. That way, at each time t a new word is added to the translated sequence and all the probabilities are computed. Only the B best possibilities are kept and expanded in the next time step.

The actual implementation for this project is a modified version of the code provided by Tensorflow in its tutorial about Sequence-to-Sequences Models [10] and will be available in a Github repository[1] under the accounts of the authors.

3 Experiments

3.1 Dataset

Since the goal of the project is to translate between English and Spanish it is crucial to use a dataset that contains sentence aligned text in both languages. For that reason, the train and test dataset used in this project were selected from the guidelines of the ACL 2007 second workshop on statistical machine translation [11], that is the European Parliament Proceedings Parallel Corpus [4].

The Spanish-English corpus is a compressed file of 187MB that contains $2\,218\,201$ sentences and $53\,974\,751$ words in English, and $2\,123\,835$ sentences and $54\,806\,927$ words in Spanish divided in two files. For validation and test, two small subsets of 2000 sentences each are taken from the large Spanish-English corpus, as suggested by [11], leaving the remaining sentences as training data.

3.2 Experimental design

The experiment set has been designed in a way that lets us appreciate the influence of several parameters, but also to find the best possible system. Those parameters settings found to be beneficial are incorporated into the system and used in the following experiments. That way we hope to obtain the best possible model, being able to train it for longer and obtain the best performing network.

The perplexity of the systems is used as an evaluation metric in all the experiments, in order to compare which strategy works better. The underlying idea behind this metric, as explained by Martin and Jurafsky [5] is that for a sequence of words, the perplexity is computed over a random variable that ranges over the sequence as $2^{H(w_1,\dots w_T)} = 2^{-\sum_{t=1}^{T} p(w_t)\log p(w_t)}$.

However, this metric should be normalized by the length of the sequence. Intuitively, a perplexity of 5 would mean that the system is as *perplexed* as if it had to choose the next word between 5 equally probable words. Technically, the perplexity is the "weighted average number of choices a random variable has to make" [5]. It is worth mentioning that the perplexity can be artificially reduced if the dataset is domain restricted.

In the following list it is possible to find the description for every designed experiment:

1. Architecture: Several architectures are tested during a small number of iterations ($10\,000$, standing for one third of an epoch given the size of the minibatches) in order to compare the perplexity of systems that maximize the number of layers, the number of hidden units per layer or systems with low model complexity.
   (a) Maximize layers: 4 layers with 256 nodes per layer
   (b) Maximize hidden units per layer: 3 layers with 512 nodes per layer
   (c) Low-performance system
2. Input order: The goal of this experiment is proving the statement made by Sutskever et al. [9] that "reversing the order of the words in all source sentences (but not target sentences) improved the LSTM's performance markedly".
   (a) Normal order
   (b) Reversed order

[1] `https://github.com/MonicaVillanueva/English_Spanish_Translator`

3. Tokenizer: In lexical analysis, tokenization is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens. The text is broken down into these pieces using a set of symbols as separators, that usually are punctuation symbols, white spaces, new lines, etc, as well as special rules that model corner cases or proper names. The list of tokens becomes input for further processing like the encoding-decoding.

   It is said in the Tensorflow [10] tutorial that improving the tokenizer function should improve the results. In these experiments the results of the basic tokenizer are compared with those of the improved version in order to perceive the level of enhancement in the translation.

   (a) Naive tokenizer
   (b) Improved tokenizer

4. Optimizer and parameters: Another source of improvements lies in the usage of more complex optimizers and the fine tuning of the network parameters. Each of the experiments is performed with three different learning rates in order to find a good setting.

   (a) GradientDescentOptimizer: "Performs an update for every mini-batch [...] reducing the variance of the parameter updates and [...] allowing the use of highly optimized matrix optimizations."
   (b) AdagradOptimizer: "It adapts the learning rate to the parameters, performing larger updates for infrequent and smaller updates for frequent parameters. For this reason, it is well-suited for dealing with sparse data"
   (c) AdamOptimizer: "Seeks to reduce [Adagrad's] aggressive, monotonically decreasing learning rate. [...] It keeps track of the estimates of the first moment (the mean) and the second moment (the uncentered variance) of the gradients." Ruder [8]

5. Vocabulary size: Increasing the size of the vocabulary is supposed to improve the quality of the translation. Different size of vocabularies are used to assess this theory.

   (a) Vocabulary of $20\,000$ words
   (b) Vocabulary of $40\,000$ words
   (c) Vocabulary of $60\,000$ words

6. Longer training stage for the best performing network. The BLEU score will be calculated for this experiment allowing a comparison between the trained system and the results in the original paper [9].

   BLEU score is a metric based on the word error rate metric, modified so that it accepts several references and sensible word variations. However, in this case, there is only one reference for each translation.

   This modification prevents the artificial high score of translations that use words that are likely to appear in the reference sentence, by clipping the total number of words in the translation by the maximum number of times that the word appears in the reference. To compute the global score, it is necessary to calculate the geometric mean of all the sentence scores and multiply it by an exponential brevity penalty factor (see Papineni et al. [7] for more details).

3.3 Hardware description and issues

The training is performed using Google's Tensorflow library, which runs in Python. The system is run in a high performance node of the PDC facility, using a K80 Nvidia GPU. This card is composed of two separated GPU cores which offer 11x2 GB of graphical memory, for a total of 24GB.

With this experimental hardware setting, we run into problems when trying to learn very deep or node dense models. We were unable to run, due to lack of memory issues, networks with 512 nodes per layer, as well as models with more than 4 layers, even when the number of nodes was as low as 256 per layer. This problem is accentuated by the extreme unavailability and over crowdedness of the PDC nodes, not being able to have access to the appropriate resources most of the time. Having 24GB of available memory, it seems weird to us running into this kind of problems, but not being experts in the matter, we were not able to sort them out in order to run models as deep and dense as the one suggested in [9].

Certain strategies, such as reducing the size of the training vocabulary, allow us to increase the complexity of the network, but we finally decided to use the predetermined size of $40\,000$ words and train the most complex model to be a network of 4 layers, using 256 nodes per layer and $40\,000$ words of both input and output vocabulary. Most of the late experiments, where certain potential improvements are tested use this network.

4 Results

4.1 Experiments evaluation

Due to the hardware resources related problem mentioned in Subsection 3.3, we have not able to run all the experiments we would have liked to using the suitable network configurations. Therefore, we decided to test our improvements in a smaller network (that can be run with inferior resources) and then apply these improvements to a final long run in the larger network. The smaller network used to test the improvements is conformed by 3 layers with 128 nodes per layer, while the last long run was carried out with a 4 layers, 256 nodes network. The results for the different improvements tested are shown in Table 1.

The baseline network configuration is, as stated above, build on 3 layers with 128 nodes per layer. It uses a basic tokenizer, reversed input sentence and vocabulary size of $40\,000$ words. Training is performed using vanilla SGD, with an initial learning rate of 0.5, a exponential decay factor of 0.99 and gradient clipping.

Table 1: Perplexity results with different improvements

	Baseline	Non-reversed input	Better tokenizer	Larger vocabulary (60K Words)	Larger architecture (4 x 256)
Perplexity (after 10K iterations)	17.50	19.18	10.23	18.04	16.99

- Architecture: Having more nodes per layer, as well as more layers, should increase the capacity of abstraction of the network, allowing it to produce better translations and therefore decreasing the perplexity. However, as can be seen in 1, there is not a big difference between a 4 layer network with 256 nodes per layer and a much simpler 3 layer network with 128 nodes per layer. This may respond to the shortness of the training used. At only $10\,000$ iterations, the two networks may still be insufficiently trained to allow their performances to diverge.
- Input order: We can see in Table 1 that the perplexity increases when the input sentence is not reversed in comparison with reversed input sentence. Therefore it becomes clear that as said by the authors of the paper, reversing the input sentence helps improving the performance of the network.
- Tokenizer: The results of the perplexity using the basic tokenizer were compared to those obtained using a better tokenizer (Penn Treebank tokenizer).

  It is possible to observe in Table 1 that the perplexity, as suggested by the tutorial authors, decreases when a better tokenizer is used. A simple example can explain this:

  Suppose that an input sentence contains the word “o’clock". The basic tokenizer would split it in three parts “o", “’" and “clock". But rather, this forms a single entity and should be tokenized as “o’clock". That is what a better tokenizer does, taking into account corner cases and other special situations. The translations would be better then when using a better tokenizer.
- Optimizer and parameters: Due to resources and time limitations this experiment is left for future work.
- Vocabulary size: Even though the perplexity has also been computed for a larger vocabulary size to get an idea of the performance (Table 1), it is not possible to compare it with the baseline, since the perplexity itself depends on the size of the vocabulary. If the vocabulary is larger, there are more words to choose from, and the system will always be more perplexed.

After noticing this fact, for this specific experiment the BLEU score should have been used, but due to the lack of resources in the PDC, only one long run (145K iterations) was performed, and for that one the size of vocabulary was the default of 40K words.

- Longer training stage with the best performing network: The final objective of this project would be to perform a longer training stage with the best performing network, using the tested improvements.
  The only real improvement tested is the tokenizer. The best performing network using this improvement has not been run due to hardware related problems.

Right now, the best performing network's perplexity value is 3.9 using 4 layers with 256 nodes and training for $145\,000$ iterations, which stands for ≈ 4.5 epochs. This perplexity is quite low, since it means that for each word, the network has an equally probable guess between only four words. However, this value cannot easily be compared to other values in the literature, because it depends heavily on the dataset. If the dataset has a restricted topic, as it is case with the European Parliament, the perplexity can be artificially low, because the prediction of next words is easier than it would be in general topic related dataset.

With this longer trained network, we translated the test set and computed the BLEU score for it. Averaging the score with the formula in Papineni et al. [7] we get that the translation achieves a score of 7.67. However, it is also recommended to tokenize the two sets to compare. That way it is possible to avoid format differences, improving the score. Applying the tokenizer used, the score increases to 10.98.

Taking a close look the origin of this score, it is possible to learn some things. With an 1-gram, that is, using one word, the score is 34.5. With a 2-gram, 15.2; 3-gram 7.3 and 4-gram 3.8. This shows that though the whole sentences may be bad translated, in quite some cases, they use words that are appropriate. This phenomenon can be related with the close domain of the dataset, or maybe, the translation process is starting to learn something significant, but it is not perfect yet.

4.2 Translation examples

From the 2000 translated sentences of the test set, we select some (good and bad) generated translations along with the reference sentence and a "translation back" into English so that non Spanish speakers can get an idea of how good the translation is. Those translations are shown in Tables 2, 3 and 4.

Table 2: Translation 1 from English to Spanish

	Reference Sentence	**Generated Sentence** *(*) translated back*
Spanish	Mi informe avala este criterio	, este informa apoya, en mi opinión, la
English	My report supports this	This report supports, in my opinion, the (*)

Table 3: Translation 2 from English to Spanish

	Reference Sentence	**Generated Sentence** *(*) translated back*
Spanish	El problema más difícil con el que se enfrenta la nueva Carta no es su contenido sino el saber si tiene o no tiene valor jurídico.	El Consejo considera que , si se trata de medio ambiente , la cuestión actual de la crisis se enfrenta a una cuestión más compleja
English	The most difficult problem facing this new charter is not its actual content but whether it has legal status.	The Council considers that, in the case of the environment, the current issue of the crisis faces a more complete question (*)

As native speakers, we can notice that the translations are quite bad in general, but coherent. They constitute well-formed meaningful sentences though their meaning is not closely related to the meaning of the input sentence. A possible reason for that is developed in Section 5. Also, it can be noticed how the system performs worse on longer sentences, as it could be expected.

Table 4: Translation 3 from English to Spanish

	Reference Sentence	**Generated Sentence** *(*) translated back*
Spanish	Así, hemos estudiado el texto y hemos intentado acortar algo una serie de puntos.	De acuerdo con esto , hemos vuelto a examinar la cuestión .
English	This is how we came to go through the text, and attempted to cut down on certain items in the process.	Accordingly, we have re-examined the issue (*)

5 Discussion and Conclusions

It is clear that the results obtained in this project are susceptible of improvement. The huge differences with Sutskever et al. [9] in terms of network size, due to hardware related issues, as well the impossibility of experimenting in depth on the effect of potential improvements, are possible reasons for this lack of performance.

In the very few experiments that have been possible to run, it is clear how the utilization of a better tokenizer goes a long way in improving the perplexity measurements of the network, and so one of the first steps to be considered as future work would be to incorporate this technique, as well as possibly other potential improvements, into the best performing network.

The current performance of the network, however, is not bad in the sense that the translations are incoherent phrases, but rather in that it produces sensible outputs that in a lot of occasions do not resemble the meaning of the original phrase partially or completely. This phenomenon, from our understanding, could be related to the sequence to sequence modeling, instead of a word to word translation approach.

In sequence to sequence translation, the system is able to learn which words have to follow some certain word, so coherence within the sentence is boosted, but it could be more difficult to learn how to relate this words to the words in the input language. On the other hand, with phrase-based translation systems, the system is able to learn correspondences between two words in different languages, but it becomes more difficult to keep the general coherence of the phrase, specially if both languages are grammatically different, which is not the case for Spanish and English, though.

References

[1] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[2] Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

[3] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[4] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, volume 5, pages 79–86. Citeseer, 2005.

[5] James H Martin and Daniel Jurafsky. Speech and language processing. *International Edition*, pages 221–228, 1999.

[6] Christopher Olah. Understanding LSTM networks. URL `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`.

[7] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. BLEU: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics, 2002.

[8] Sebastian Ruder. An overview of gradient descent optimization algorithms. URL `http://sebastianruder.com/optimizing-gradient-descent/index.html`.

[9] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[10] Tensorflow. Sequence-to-sequence models | tensorflow. URL `https://www.tensorflow.org/tutorials/seq2seq`.

[11] Statistical Machine Translation. Workshop shared task: Machine translation for european languages. URL `http://www.statmt.org/wmt07/shared-task.html`.

[12] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

[13] Heiga Ze, Andrew Senior, and Mike Schuster. Statistical parametric speech synthesis using deep neural networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 7962–7966. IEEE, 2013.