Learning Constraint Rankings with Sequence to Sequence Models

Tovly Deutsch

Ling 115

1 Introduction

Phonological systems are often accompanied by algorithms used to derive the applicable processes, e.g. constraints or rules, for a language or example. Optimally Theory (Prince and Smolensky 1993) is one such phonological system involving constraints on phonological patterns that are ranked with respect to one another. A variety of learning algorithms have been proposed for Optimally Theory (Tesar and Smolensky 2000; Boersma and Hayes 1999; Doyle, Bicknell, and Levy 2014). These algorithms are usually framed in the context of selecting some examples that illustrate a phonological pattern, selecting some constraints that may be appropriate for the pattern, and then running the algorithm to rank those constraints. However, these algorithms often make certain assumptions about the underlying language that may not be correct or determinable. For instance, take recursive constraint demotion (Tesar and Smolensky 2000), a relatively simple algorithm for learning constraint rankings. This algorithm fails to account for variation in the data. Natural variation allows winning candidates to vary among speakers indicating different constraint rankings. Additionally, the algorithm requires as input the underlying representations (URs) of the candidates. These URs may be difficult or time-consuming to determine or may even be posited to not exist as a natural aspect of speakers' phonological knowledge.

These shortcomings motivate the use of a less rigid algorithm for learning constraint rankings. One approach is to base the phonological model or learning algorithm on statistical methods to allow it to account for the variation observed in phonological data. Stochastic Optimality Theory (Boersma 1970) attempts to account for variation by describing the rank of each constraint as being drawn from a normal distribution. However, the framework still requires some learning algorithm, like the gradual learning algorithm (GLA) (Boersma and Hayes 1999), that requires access to posited underlying representations. Given the desire for underspecifying the input and extracting complex phonological patterns from such input, machine learning methods may be appropriate for this type of task. Many tasks in natural language processing use machine learning models to infer outputs from patterns in input data, the same paradigm observed in learning constraint rankings from language examples. In particular, the recently popular sequence to sequence models (Sutskever, Vinyals, and Le 2014) are especially well suited to learning constraint rankings. These models, originally created for natural language translation, are trained to receive an input sequence of one type and output a sequence of another type. In the context of constraint ranking learning, the input sequence could be the phonological examples from the language and the output examples could be the ranked sequence of constraints. Additionally, techniques like beam search (Graves 2012) allow such models to output multiple hypotheses each marked with a probability of correctness. These multiple hypotheses can thus account for the variation present in the language. This paper describes a small scale prototype of this approach for learning constraint rankings, trained and evaluated on a small subset of artificially constructed English-like examples. Doyle, Bicknell, and Levy (2014) take a similar approach in using a Bayesian model to learn constraint rankings. However, their work fails to incorporate variation into the input data, a key component of the experiments presented in this paper. While the results of this paper show some promise in learning basic constraint rankings, the models thus far tested are significantly limited in their overall accuracy and ability to account for variation.

2 Methodology

Ultimately, phonological learning algorithms should learn some constraint rankings given some samples from a language. This process can be modeled as a sequence to sequence task, a type of transformation well studied in

Segment type	Segments
All segments	æ, ɔ, ɛ, e, I, k, g, d, h, t, ʔ, w, s, z, n, l, v, f, θ
Vowels	æ, ɔ, ɛ, e, I
Consonants	k, g, d, h, t, ʔ, w, s, z, n, l, v, f, θ
Voiced consonants	d, w, g, z, v, n, l
Voiceless consonants	k, h, t, ʔ, w, s, f, θ
Sonorants	æ, ɔ, ɛ, e, I, n, l, w
Obstruents	k, g, d, h, t, ʔ,s, z, v, f, θ
Voiced Obstruents	g, d, z, v
Voiceless Obstruents	k, h, t, ʔ, w, s, f, θ

Table 1: The segments used to construct the artifical English-like dataset.

natural language processing. I frame phonological learning as a task of mapping a sequence of sample words to a sequence of constraints, output in order from highest to lowest rank. To infer this constraint sequence, I use a sequence to sequence model (Sutskever, Vinyals, and Le 2014) paired with beam search (Graves 2012). This method of learning has two advantages over some other learning algorithms such as recursive constraint demotion. First, in usage on language data, the input is taken as phonological examples and thus requires no positing of underlying representation. Instead, the model must learn any possible underlying representations. Second, via techniques like beam search, the model can produce multiple output rankings with differing probability of correctness, thereby accounting for variation.

2.1 Dataset construction

To train a machine learning model, a significant amount of data is required. As a starting point for this work, I elected to construct an limited artificial dataset for this task based on English. I limited the segments available to a subset of those found in English as shown in table 1. Similarly, I limited the output to only four possible constraints related to an English final devoicing phenomenon. Tableau 1 demonstrates the use of these constraints for the English final devoicing example.

(1) *cats* Tableau

kætz	Agree	*Ident-IO(voi)	*D	*D]$_\sigma$
☞ a. kæts		$*$		
b. kætz	$*!$		$*$	$*$
c. kædz		$*$	$**!$	$*$
d. kæds	$*!$	$**$	$*$	

To construct each example, I begin by defining an abstract underlying representation (UR). This UR is abstract in that it does not specify a particular segment but rather *segment types* that are relevant for the allowed constraints. For instance, one abstract UR might be “CVTD”, where “C” is a consonant, “V” is a vowel, “T” is a voiceless obstruent, and “D” is a voiced obstruent. Given this underlying representation, I then construct a variety of *abstract surface forms*, also segment type sequences, that could be possible abstract surface forms for the UR given some ranking of the constraints. The underlying representations and surface forms are made abstract, and ultimately variable, to provide irrelevant variation (noise) that the model must parse through, as it would for real data. These abstract surface forms are paired with their corresponding constraint rankings to form *abstract examples*. Finally, given these abstract examples, concrete examples are constructed for the dataset by selecting random segments within each segment type. For each abstract example, a random number of concrete examples are drawn. For each concrete example drawn, a random number of concrete surface forms are generated based on the abstract surface form. Within each concrete example, the concrete surface forms are separated by a special separator token to indicate word boundaries. Some examples of these concrete examples are shown in table 2. For all model training and evaluation, the model is never provided abstract URs or abstract surface forms, only the concretely drawn samples.

Abstract UR	Abstract surface form	Sampled surface forms	Constraint ranking
CVTD	CVTD	wesz, gɔθd, zIθg, lɛʔg	Ident-IO(voi) » Agree » *D » *D_σ
CVTD	CVTD	tɔkz, defd, geθg, netv	Ident-IO(voi) » Agree » *D » *D_σ
CVTD	CVTT	kɛtθ, ʔɔtf, gætθ, ʔɛtθ	Agree » Ident-IO(voi) » *D » *D_σ
CVTD	CVTT	tætθ, kɔtk, wækθ, zɛff	Agree » Ident-IO(voi) » *D » *D_σ
DTVD	TTVT	kteh, wsɔk, tθæk, ʔtɛθ	*D » Agree » Ident-IO(voi) » *D_σ

Table 2: Selected concrete examples.

2.2 Model

A popular type of model in natural language processing is known as the sequence to sequence model (Sutskever, Vinyals, and Le 2014) which aims to take in an input sequence of some form and produce a corresponding output sequence. Initially, this type of model was created for the purpose of translation but has since been applied to a variety of other tasks like text generation and question answering (Devlin et al. 2019). Often, a sequence to sequence model is composed of two interacting models, an encoder and a decoder. The encoder ingests the input sequence and produces a hidden representation of that input. This hidden representation is thought to encode the features of the input most salient for the decoding process. Then, this hidden representation is fed to the decoder which uses it to produce an output sequence. If the output sequence is of unbounded length, the decoder can output tokens one at a time until such a time that it determines the sequence should be completed, at which point it can output a special end of sequence (EOS) token to terminate the decoding process. Figure 1 demonstrates this process with German to English translation.

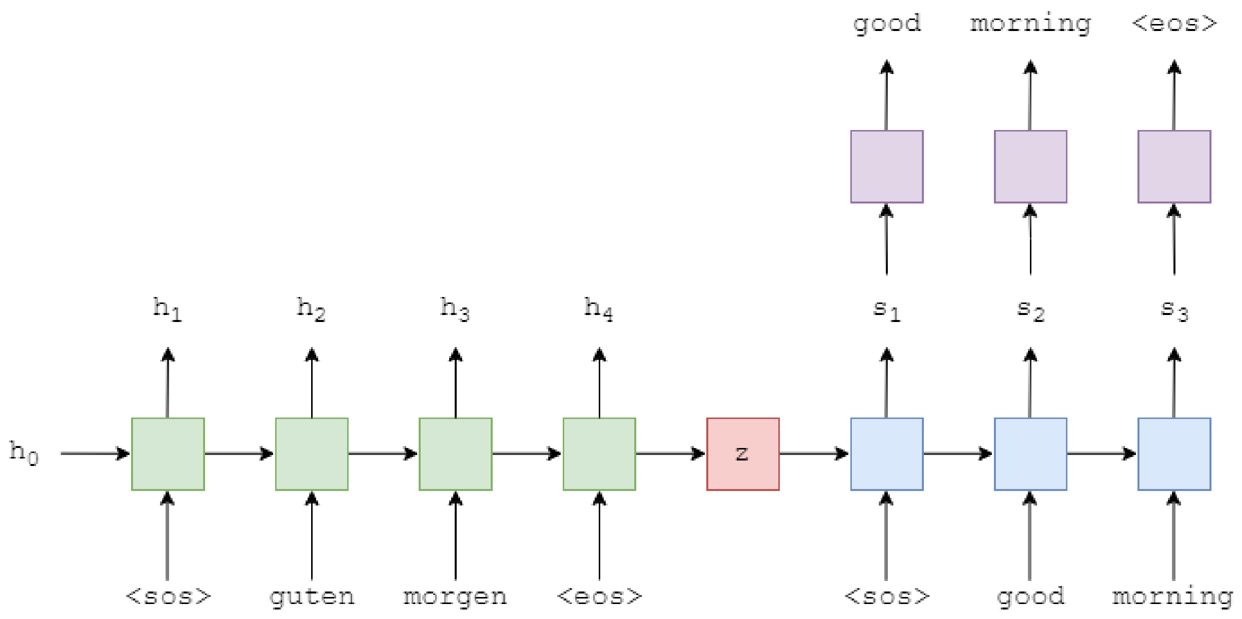

Figure 1: Diagram illustrating the overarching structure of sequence to sequence models, from Trevett (2018).

Usually, the encoder and decoder models are some type of recurrent neural network. Neural networks have become especially popular for a variety of natural language processing tasks for their ability to model complex relationships learned from large datasets. They involve neurons, modeled after human neurons, passing values between each other via linear transformation functions and non-linear activation functions. Recurrent neural networks are specifically suited to sequence tasks because they can accept inputs of unbounded length, one token at a time.

For the model in this paper [1], I utilized a specific type of RNN, an LSTM, as it has been shown to perform

[1]The models used are based on those from the pytorch-seq2seq models found at https://github.com/bentrevett/pytorch-seq2seq. The code used for the experiments in this paper can be found at https://github.com/TovlyDeutsch/115FinalProject.

better than standard RNNs at modeling long term dependencies in sequences. The encoder and decoder are each composed of an embedding layer that maps input tokens to dense vectors of length 256. These vectors are then fed into a 2 layer LSTM with a layer size (number of neurons) of 512. A dropout layer is also added to regularize the model which randomly ignores (sets to zero) half of the input values in order to prevent the model from becoming overfitted to the training data. Where not specified, the model is trained for 20 epochs, 20 complete passes over the training data. Most of these parameters where selected based on the existing models and their seemingly good performance for existing tasks. The number of epochs was chosen via a few trials as a balance between the amount of training time and model performance.

2.2.1 Beam Search

In the typical decoding process of sequence to sequence models, the decoder outputs the mostly likely token at each time step. This approach is known as a greedy decoding. An alternative approach, known as beam search, keeps k of the most probable sequences along with their probabilities of correctness. In the context of generating constraint rankings, this may account for variation. This ability is important given standard Optimality Theory and learning algorithms are unable to account for the variation seen in language. The output probabilities could be interpreted as the probability that any given speaker has that constraint ranking or that any given utterance will adhere to that constraint ranking. This approach is limited in that a fixed number of possibilities must be output for every prediction, even when more or fewer constraint rankings may be appropriate. An overproduction of outputs can be mitigated by encouraging the model to output zero or near zero probabilities for implausible results. Underproduction of outputs could be mitigated by increasing the number of outputs to the point where it is extremely unlikely that a language or phonological process exhibits that many variants.

2.3 Evaluation Metrics

For each output token, the decoder of the sequence to sequence model outputs a distribution over all possible tokens in the output vocabulary. Given this distribution, one way to measure model performance is to analyze how much the true distribution differs from this distribution. One method of doing this is known as cross-entropy loss, presented in equation 1. To calculate cross-entropy loss, for each class c (possible token) of M classes, the probability from the true distribution t is multiplied by the log of the probability from predicted distribution p. Then, these M products are summed. Because only the true class will have a probability of 1 in the true distribution and all other classes will have 0 probability in the true distribution, only that class will effect the result. If the predicted probability is larger for that class, the loss will increase indicating a better result. The log is taken so that the loss is bounded above (the best result is a probability of 1 for the true class) but unbounded below (the log(x) approaches negative infinity as x approaches 0) indicating any degree of error. Often, the cross entropy loss is negated so that a larger value indicates more error as optimization methods tend to be constructed to minimize rather than maximize the loss function. This final form of cross-entropy loss is presented in equation 1. Because cross entropy is calculated for each output token, to calculate cross entropy for a full example the cross entropy for each token is summed. Perplexity, another measure for sequence model performance, is defined as $2^{\text{cross entropy}}$ and often provided in addition to or instead of cross entropy as a more interpretable measure. Lower cross-entropy loss and perplexity both indicate greater performance

$$L = -\sum_{c=1}^{M} t(c)\log(p(c)) \tag{1}$$

3 Results

Overall, the model was able to learn to associate sensible constraint rankings for some examples. Often, it constrained itself to a limited set of possible rankings and pathologically preferred certain rankings regardless of their correctness. Additionally, the model struggled to generalize well to word type ranking associations it had not observed during training. The initial results for the model trained and tested on all examples are presented in table 3. Select test examples are presented in table 4. From the given examples, it is clear that the model prefers to have faithfulness constraints have the highest rank despite the fact that the training data was not biased toward this type of ranking.

Perhaps this occurs because the model fails to make the same underlying representation assumptions that I did and thus simply assumes that the forms given are the underlying representations, even when these assumptions are incompatible across samples.

Test set	Number of training epochs	Loss (Cross Entropy)	Perplexity
All examples	1	0.809	2.247
All examples	20	0.396	1.486
Double vowel examples	20	1.043	2.873

Table 3: Evaluations for model on test sets.

In addition to testing the model on a dataset of all the example types discussed in section 2.1, I also performed experiments in which I held out a specific example type (examples with two vowels) from the training set and composed the test set entirely of that example type. This experiment assesses the model's ability to generalize to new example types it has not observed during training. As shown in table 3, the model does this poorly. The specific dataset tests (the double vowel examples) have much larger losses and perplexities, often double to triple the metrics of the standard test. This is not entirely unsurprising giving that generalization is one of the tasks that complex machine learning models struggle to perform well. Usually, one remedy to this issue is having the model train on a larger and more diverse set of data. This approach is especially promising for future work considering the small size and few example types for the dataset constructed in this paper.

UR	Abstract surface form	Model input	Output ranking	Correct
DTV	DTV	ghI, vθe, zsɔ, gʔɔ	*Ident-IO(voi) » Agree » *D » *D$_\sigma$	True
CVtθ	CVtθ	θɔtθ, fætθ, θɔtθ, gɔtθ	*Ident-IO(voi) » Agree » *D » *D$_\sigma$	True
CVTD	CVTD	wesz, gɔθd, zIθg, lɛʔg	*Ident-IO(voi) » Agree » *D » *D$_\sigma$	True
CCVlvθ	CCVlvθ	θʔɔlvθ, hkɔlvθ, fkɛlvθ, lhIlvθ	*Ident-IO(voi) » Agree » *D » *D$_\sigma$	True
CCVlvθ	CCVlvθ	hhelvθ, θhIlvθ, lhelvθ, klIlvθ	Agree » *Ident-IO(voi) » *D » *D$_\sigma$	False
DTV	TTV	fθI, tkI, kθæ, θhɛ	Agree » *Ident-IO(voi) » *D » *D$_\sigma$	True
CVtθ	CVtθ	tɛtθ, ʔɔtθ, gætθ, ʔɛtθ	Agree » *Ident-IO(voi) » *D » *D$_\sigma$	True
VDTV	VTTV	ɛkhɛ, ɛtʔæ, Ifʔe, ɛhkI	*D » Agree » *Ident-IO(voi) » *D$_\sigma$	True

Table 4: Test examples and results for model trained on all example types.

Looking at specific model evaluations of the double vowel example types in table 5, it is clear the model still has many basic facts about OT to learn considering it often repeated constraints, a phenomenon it had not observed at all in the training set. On the other hand, the fact that it is outputting rankings it had not seen before indicates that it may have some knowledge that this type of input is novel and thus requires a new type of output. This result shows promise for the model to generalize at some point, perhaps given more data or a more complex model type.

UR	Abstract surface form	Model input	Output ranking	Correct
VDTV	VTTV	ɛkʔɔ, ɛtfɛ, IttI, ɔfθI	Agree » Agree » *D » *D$_\sigma$	True
VDTV	VTTV	ɛtsɔ, Ihte, eʔθɛ, æhʔæ	Agree » Agree » *D » *D$_\sigma$	True[2]
VDTV	VTTV	efsæ, æthɔ, ɔθhæ, æffɔ	Agree » *Ident-IO(voi) » *D » *D$_\sigma$	True

Table 5: Test examples with two vowels in the UR.

3.1 Beam Search

Given that there may be variation in the input data, I used a beam search decoder in order to produce various possible constraint rankings along with their probabilities of correctness. I selected beam search outputs for which the top two most probable decoded sequences (constraint rankings) differ in probability by at least 0.1, indicating that the model has some confidence about one ranking being more probable than the other. A few beam search output examples are presented in table 6. Despite its promise, the output of the beam search decoder was rarely accurate.

[2]Although this ranking technically captures the phenomena of devoicing, it is illogical in repeating a constraint.

There was a tendency for repeated constraints and rankings with length different from four, both phenomena that were not present in the input data.

UR	Abstract surface form	Model input	Confidence probability	Output ranking
CCVlvθ	Ç̥Ç̥Vlfθ	sfIlfθ, sfIlfθ, kθɔlfθ, fkelfθ	0.170	*D$_\sigma$ » *D$_\sigma$
CCVlvθ	Ç̥Ç̥Vlfθ	sfIlfθ, sfIlfθ, kθɔlfθ, fkelfθ	0.067	*D » *Ident-IO(voi) » *D$_\sigma$ » *D$_\sigma$
CVnz	CVnz	kænz, nInz, fenz, tɔnz	0.192	*Ident-IO(voi)
CVnz	CVnz	kænz, nInz, fenz, tɔnz	0.105	*D » *Ident-IO(voi) » *Ident-IO(voi)

Table 6: Example outputs of beam search.

3.2 Effects of Data Size

Because the dataset is stochastically generated with some parameters, as described in section section 2.1, these parameters can be modified to observe how the model handles different formulations of the dataset. The two parameter sets available are the minimum and maximum number of samples per abstract example and the minimum and maximum number of samples per abstract surface form.

3.2.1 Manipulating Samples Per Abstract Surface Form

As shown in table 7, initially, increasing the number of samples per abstract surface form appears to have a significant positive effect on model performance. Holding the minimum number of samples at 1 and increasing the maximum number of samples from 1 to 5, the loss and perplexity roughly improve by a factor of 2. This performance increase supports the hypothesis that multiple samples illustrating the same phonological pattern will better allow a model to understand what segments of the input to focus on and which segments may be irrelevant for the particular phonological process. At a certain point, increasing the number of samples seems to have a slight negative effect on the performance of the model. Both the loss and perplexity increased by about 10% when the minimum and maximum number of samples were increased by an additional order of magnitude. The information shared between the word samples are largely the same indicating one form of voicing placement. The primary purpose of multiple samples is simply to teach the model which portions of the word may be irrelevant to determining the constraint ranking. Thus, it is likely the case that after a certain number of examples, the increase in the number of samples becomes redundant noise that confuses the model and reduces its performance.

Min samples per abstract surface form	Max samples per abstract surface form	Loss (cross entropy)	Perplexity
1	1	0.952	2.592
1	5	0.396	1.486
10	50	0.463	1.589

Table 7: Evaluations for model on test set of all examples with varying ranges of samples per abstract surface form. The number of samples per abstract example ranges between 10 and 100.

3.2.2 Manipulating Samples Abstract Example

As with varying the number of samples per word abstract surface form, varying the number of samples per abstract example can have significant effects on model performance. Beginning with a baseline of between 10 and 100 samples per abstract example, decrementing the range limits by one order of magnitude to between 1 and 10 samples has little to no effect on model performance as shown in table 8. By contrast, incrementing the range limits by an order of magnitude worsens model performance by approximately a factor of two. Perhaps this occurs because the number of epochs, the number of passes over the training data, is fixed between tests; in the test with a larger amount of training data, this fixed number of epochs may be insufficient to learn the patterns present in the data.

Min samples per abstract example	Max samples per abstract example	Loss (cross entropy)	Perplexity
1	10	0.465	1.592
10	100	0.396	1.486
100	1000	0.854	2.349

Table 8: Evaluations for model on test set of all examples with varying ranges of samples per abstract example. The number of samples per abstract surface form ranges between 1 and 5.

4 Conclusion

The model tested in this paper had some success in predicting constraint rankings. However, its frequent failures along with the simplicity and small scale of the data tested indicated that much improvement can still be made to the model's performance. Such improvement may be found in exploring other model types like more recent advancements in sequence to sequence models.

Similarly, attempts to account for variation via beam search largely failed. Perhaps other techniques to produce multiple hypotheses may be appropriate. One possible method is to train multiple models, one for each variant of a proposed ranking and use these as proxies for differing groups of speakers. Other possible methods might include tagging output data in the dataset with the probabilities with which they are observed. This is in contrast to the technique used in this paper in which output frequencies are simply indicated by the proportions of the examples, not within each example itself.

An additional limitation of this modeling approach is overgeneration. Specifically, the ability of the model to output multiple variational hypotheses allows it to posit radically different phonological processes for different speakers within one language, e.g. two rankings that are complete reversals of one another. Such power may be undesired in describing real languages. However given enough data, it is likely that these models would avoid generating such extreme outputs.

Finally, this work is limited in predicting constraint rankings solely in the Optimality Theory framework. Future work could attempt to train models to produce phonological constraints in other frameworks like Harmonic Grammar (Legendre, Miyata, and Smolensky 1990), Stochastic OT (Boersma 1970), and Serial Harmonic Grammar (Pater 2010) given the descriptive advantages such frameworks may provide.

Thus, a machine learning sequence to sequence approach for learning patterns from phonological data may be valid. However, significant limitations in the current approach must be overcome for practical use in analysis. Such use may allow these models to overcome the limitations imposed by existing learning algorithms and phonological frameworks .

References

Boersma, Paul (Feb. 1970). “How We Learn Variation, Optionality, And Probability”. In: *Proceedings of the Institute of Phonetic Sciences Amsterdam* 21.

Boersma, Paul and Bruce Hayes (1999). “Empirical tests of the gradual learning algorithm”. In: *Linguistic Inquiry* 32, pp. 45–86.

Devlin, Jacob et al. (May 2019). “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”. In: *arXiv:1810.04805 [cs]*. URL: http://arxiv.org/abs/1810.04805 (visited on 12/09/2019).

Doyle, Gabriel, Klinton Bicknell, and Roger Levy (June 2014). “Nonparametric Learning of Phonological Constraints in Optimality Theory”. In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Baltimore, Maryland: Association for Computational Linguistics, pp. 1094–1103. DOI: 10.3115/v1/P14-1103. URL: https://www.aclweb.org/anthology/P14-1103 (visited on 12/09/2019).

Graves, Alex (Nov. 2012). “Sequence Transduction with Recurrent Neural Networks”. en. In: URL: https://arxiv.org/abs/1211.3711v1 (visited on 12/09/2019).

Legendre, Geraldine, Yoshiro Miyata, and Paul Smolensky (July 1990). “Can Connectionism Contribute to Syntax? Harmonic Grammar, with an Application ; CU-CS-485-90”. In: *Computer Science Technical Reports*. URL: https://scholar.colorado.edu/csci_techreports/467.

Pater, Joe (2010). “Serial Harmonic Grammar and Berber Syllabification”. In:

Prince, Alan and Paul Smolensky (Apr. 1993). “Optimality Theory: Constraint interaction in generative grammar”. URL: http://roa.rutgers.edu/files/537-0802/537-0802-PRINCE-0-0.PDF.

Sutskever, Ilya, Oriol Vinyals, and Quoc V. Le (Dec. 2014). “Sequence to Sequence Learning with Neural Networks”. In: *arXiv:1409.3215 [cs]*. URL: http://arxiv.org/abs/1409.3215 (visited on 12/03/2019).

Tesar, Bruce and Paul Smolensky (Sept. 2000). *Learnability in optimality theory*. Cambridge, MA: MIT Press. URL: https://www.researchgate.net/publication/2459518_Learnability_in_Optimality_Theory_long_version.

Trevett, Ben (Sept. 2018). *Sequence to Sequence Learning with Neural Networks*. URL: https://github.com/bentrevett/pytorch-seq2seq/blob/master/1%20-%20Sequence%20to%20Sequence%20Learning%20with%20Neural%20Networks.ipynb (visited on 12/09/2019).