Binârî: A Poetry Generation System for Ghazals

Galip Ümit Yolcu

2016400249

Supervisor: Tunga Güngör

Final report for
CmpE 492 Senior Project

Department of Computer Engineering
Faculty of Engineering
Boğaziçi University

1 July 2020

Abstract

Poetry generation is in the intersection of Natural Language Processing and Computational Creativity. It has been studied intensively from different perspectives: poetry as translation, summary, prediction, constraint satisfaction, information retrieval etc. As a result, many different tools and techniques have been used. Techniques for solving other problems that can be formulated in these perspectives have been modified to meet the poetry task and original techniques have been created.

In this study, we will present Binârî, a poetry generation system for the ghazal genre. Ghazals originated in about 7^{th} century in Arabic poetry. It then spread to other cultures and has been used in Persian, Urdu and Ottoman poetry across centuries. In our study, we will try to generate ghazals in Ottoman Turkish. This task has some difficulties that are not poetry related, regarding lack of datasets and raw data. It also has difficulties in terms of poetry generation: ghazal is a highly restrictive genre, having constraints in terms of structure, rhyme, rhythm, content and tone.

Acknowledgements

Many people were interested in this project and have made important contributions. These are friends and professors from a wide range of backgrounds that shared interest in the subject and provided me with valuable inputs in many different aspects. Of course, we must also not forget the people that don't know me, but whose work this project depends on. I feel the need to thank them all for making the process easier, more fun, and possible.

I had great help from N. İpek Hüner Cora regarding Ottoman Turkish. She has taught me the fundamentals of the subject and contributed immensely by pointing me to the data I needed, which is a major prerequisite for this kind of research.

Furthermore, I want to thank my advisor, Tunga Güngör, for the time and energy he has put in for the project, the discussions we have had, but even more than that, for the guidance he has given when I didn't know how to proceed.

I have also received great support from our department's dear professor Suzan Üsküdarlı and my valuable friend Oğuz Kaan Yüksel during the entire process. We had very exciting and fruitful conversations and they have bombarded me with insightful ideas when I needed a push.

Contents

CHAPTER 1

Introduction

1.1 Poetry Generation

Poetry generation can be seen as a subfield of Computational Creativity, defined by Colton and Wiggins [**1**] as

> "*The philosophy, science and engineering of computational systems which, by taking on particular responsibilities, exhibit behaviours that unbiased observers would deem to be creative.*"

This definition includes endeavours of creating music, paintings, amd even theorems. Poetry generation is in the intersection of Computational Creativity and Natural Language Generation, along with computational humor, song lyrics generation, story generation etc. Thus, it is also tightly connected to Natural Language Processing (NLP) and techniques from NLP and in general AI are constantly used.

The very first efforts in computer generated poetry came from the French organisation ALAMO [1]. Their most known application is called *Rimbaudelaires* which is a combinatorial program that generates sonnets by placing words taken from Charles Baudelaire's poems in templates generated from Arthur Rimbaud's work. Examples can be seen in their website. Prior to ALAMO, the OuLiPo[2] movement consisting of French mathematicians and writers suggested that constraints are a way into creating better literary texts. A very celebrated example is Georges Perec's *La disparition* where he completes the novel without using the vowel "e"(a very impressive thing to do in a language in which the word "I" is "Je"). This group was founded in 1960 and it tried to find some automatic processes of literary text generation too. For example, one of the founders of the group, Raymond Queneau wrote a book of 10 sonnets where every sonnet obeys the same rhyme scheme and sounds thus by interchanging the lines in the poems, 10^{14} poems can be generated.

Though historically valuable, these movements were essentially movements of art and they have had little contribution to the science and engineering of poetry generation systems. The first works on this area by computer scientists have appeared after the year 2000. [**2**]

In terms of scientific research, poetry generation has advantages and disadvantages. As [**1**] suggests, validation of outputs and evaluation of systems pose a great challenge in most creative systems. Poem generators are no exception. This is because there is no clear definition

[1] l'Atelier de Littérature Assisté par la Mathématique et les Ordinateurs (Atelier of Literature Assisted by Mathematics and Computers) Website: http://www.alamo.free.fr

[2] Ouvrier de Littérature Potentielle (roughly translated as Workshop of Potential Literature)

of what a poem is. On what conditions can a sequence of words be recognised as a poem? What are the conditions for it to be judged as a good one? These questions can not be addressed formally and precisely.

It is true that most poetry forms have a set of well defined constraints that must be met (for example in sonnets). These could be

(1) **Formal constraints**: Sonnets consist of 14 lines (3 quatrains and a couplet)
(2) **Rhyming constraints**: Sonnets follow the rhyme scheme ABAB | CDCD | EFEF | GG
(3) **Coherence**: Poems generally need to be semantically coherent, although there are movements that reject this

These constraints are easily checked and let us label a large portion of possible texts as not poems. However, by their nature, poems don't have a clear and unambiguous message. It is often possible that two people ascribe different meanings to a poetic sentence, or that they disagree severely on the quality of a given poem. Moreover, use of creative similes, metaphores and figurative discourse suggest that poems include phrases that are not found in the context of standard daily language and it is not straightforward to determine which phrases are allowed. In addition, some syntactical rules of language may be violated in order to enhance meaning, for example by inverted or incomplete sentences.

In contrast, this ambiguity in meaning and may in turn be an advantage for the system: it can be seen as freedom. It suggests that the semantic constraints may be relaxed and thus the task may become easier.

In his seminal work [**3**], Manurung discusses the issue of poetic license in connection with poetry evaluation and gives his definition of poetry that is used for evaluation of his system, McGonagall:

> "*According to the Oxford English Dictionary, poetic license is the 'deviation from recognized form or rule, indulged in by a writer or artist for the sake of effect', whereas the Oxford Advanced Learner's Dictionary states that it is the 'freedom to change the normal rules of language when writing verse (e.g. by reversing word order, changing meaning etc.)' The crucial question is: when appraising the output of a software program that is claimed to generate poetry, to what extent do we hold poetic license accountable for deviations that may be present in the text?*"

Manurung then argues that for a scientific study of poetry generation, the criteria should be falsifiable. This is why usage of poetic license should be kept at a minimum and it should be justified by the system. Thus, he proposes a fairly restrictive definition that contains formal, rhythmic and semantic constraints:

(1) **Grammaticality:** This constraint states that a poem must contain lines that are grammatically correct.
(2) **Meaningfulness:** Grammatical correctness does not imply that the line will make sense. This constraint says that the poem should be intentionally conveying a

conceptual message under some interpretation. This constraint limits the usage of poetic license.

(3) **Poeticness:** This constraint states that the poem must have some features that discriminate it from prose. These may be with respect to form, rhythm or rhyme.

Notice that constraints (1) and (2) should be satisfied by any text, poetic or prose, meaning no relaxation is done in terms of poetic license. Constraint (3) adds the condition of complying to a poetic form.

1.2 Ghazal Poetry

Ghazal is a poetic form in Arabic, Persian, Urdu and Ottoman poetry. In this study, we will focus on ghazals in Ottoman Turkish, though everything we say about ghazals should be -in general, because ghazals have a long history and were undeniably subject to change- true for ghazals in all languages.

1.2.1 Ottoman Turkish

The poems we will try to generate will be in Ottoman Turkish. While there are discussions on the matter, it is fairly appropriate -especially in the context of NLP- to say that this is not a seperate language from Turkish.

Mostly, the grammar is the same: some grammatical structures that are dead in modern Turkish are present in Ottoman Turkish, also some constructs(for example for noun phrases) are borrowed from Persian and Arabic.

In terms of vocabulary, Persian and Arabic words are used along with the Turkish ones.

1.2.2 Formal Constraints

Ghazals are composed of 5 to 15 couplets(a poetic device composed of two lines). Technically, they are not couplets, since most of them don't have internal rhyme. The correct word is *beyt*, but for clarity we will use the word *couplet* in this study.

The first couplet contains two lines that rhyme. After this, all following couplets' second lines have to rhyme with the first couplet. Furthermore, all lines in a ghazal need to follow a pattern according to their syllables. Each syllable is either closed (also called long, designated with the sign $-$) or open (also called short, designated with sign $\cdot$) depending on the time it takes to pronounce that syllable. The pattern of syllables in each line should conform to a selected pattern. This kind of metre is called *Aruz metre* and is generally used in Ottoman poetry.

In Turkish, there are many open syllables whereas the number of closed syllables are fairly low. Most closed syllables are in words that are borrowed from Persian and Arabic. Therefore, poets frequently go out of the metre and use open syllables ($\cdot$) instead of closed

ones (–). While this is acknowledged as a flaw, it is not a grave one, whereas doing the inverse (i.e. using closed syllables instead of open) is considered a big mistake.

Lastly, the final couplet must include the poet's pen-name.

1.2.3 Semantic Constraints

Ghazals are restricted in topic and tone too. While there are always exceptions, ghazals are about the poet's beloved one. The beloved is described, praised, magnified and even criticized by the poet. This is never done in a happy tone. The lover(poet) is always in trouble because the beloved never accepts, or generally doesn't even care about the poet. It is common that the poet talks about an adversary that tries to reach the beloved too.

1.2.4 Manurung's Evaluation Criteria with Ghazals

From the definition above, we can see that Manurung's criteria of Grammaticality, Meaningfulness and Poeticness fits nicely to our aim. Since ghazals are constrained in topic and tone, meaningfulness is both more easily achieved and tested. Also, even though ghazals allow for poetic license in terms of relaxed grammatical constraints and figurative phrases, these are generally done in a standard fashion and the criteria can be altered to allow for these.

1.3 Other Poetic Forms

There are some poetic forms that have been studied more than the others in the poetry generation domain. It will be useful to summarize them here.

1.3.1 Sonnet

Sonnets are poems that follow iambic pentameter. In this metre, there are again two kinds of syllables: stressed and unstressed. Iambic pentameter is created by repeating the pattern (unstressed - stressed) 5 times. As explained above, it consists of 3 quatrains and a couplet. The lines in the couplet rhymes internally and the quatrains have ABAB rhyme scheme.

The topic varies across time, but during its peak with William Shakespeare, with whom the genre is associated because of his famous sonnets, the topic was again the poet's love for some beloved.

1.3.2 Haiku

Haikus are traditional Japanese poems. They consist of 3 short lines having 5, 7 and 5 syllables respectively. They are a form where poetic license is very freely used and the meaning is not clear at all, they leave the reader wondering. Haikus are concentrated on saying as much as possible using very few words. They aim to evoke or express emotions in or to the reader. **[4]**

1.3.3 Classical Chinese Poetry

In classical Chinese poetry, number of characters/syllables in each line is constant, 5 or 7 per line. Most of the lines should rhyme and some tonal metres should be followed. [**5**]

CHAPTER 2

Literature Review

In [**6**], it is suggested we can roughly classify poetry generation systems into 5 clusters in terms of the techniques they use:

(1) Template Based Approach
(2) Generate and Test Approach
(3) Evolutionary Approach
(4) Case Based Reasoning Approach
(5) Stochastic Language Modeling

Since we are convinced that Manurung's criteria are suitable for our task, we will take a different road (somewhat more similar to Oliveira's survey [**2**]) and discuss how the systems in the literature succeed to satisfy our 3 criteria. Then we will discuss some other evaluation strategies used by the researchers in the field.

2.1 Satisfying Grammaticality

Grammaticality is the constraint that every sentence in the poem should comply to the grammar of the language. It is almost exclusively satisfied during the generation, that is, a sentence that is not grammatically correct is never generated.

There are 4 main approaches taken to ensure this: using some grammar formalism, templates, n-gram and neural language models.

Toivanen et al. suggest another alternative: using constraint logic programming techniques to define grammar features as constraints on words. While this may be theoretically true, it is a much too tedious endeavour, it is very error-prone and hard to verify. They too have taken the approach 2.1.2.2 in their paper. [**7**]

2.1.1 Generative Grammars

Grammaticality is most surely satisfied if we only use sentences that are generated with the grammar itself. To this end, two kinds of grammar formalisms have been used.

2.1.1.1 Lexicalised Tree Adjoining Grammars

Lexicalised Tree Adjoining Grammars (LTAG) is a grammar formalism consisting of 2 kinds of elementary trees: initial trees and auxiliary trees. These trees contain nonterminals

as leaf nodes and can be combined to generate bigger trees. When the leaf nodes are anchored with terminal symbols (words), each tree becomes a phrase or sentence. Using LTAG, the system McGonagall [**3**] is able to ensure the generated sentences are grammatically correct.

For each sentence (i.e. each tree that is generated by combining the two types of elementary trees), we can generate a tree called the derivation tree. This is a tree where *the nodes represent the elementary trees* and *the edges represent the combining operations acted on these trees* to generate the final sentence. Notice that by incorporating the poem structure into the grammar, we can generate a derivation tree for the whole poem. Then, these derivation trees can be used as the genotype to initiate evolutionary dynamics(by using other constraints as the fitness function) and a stochastic search can be achieved. This is essentially what Manurung does in his PhD dissertation. [**3**]

2.1.1.2 Context Free Grammars

Tobing et al. [**8**] use chart parsing in the reverse direction, for sentence generation, using context free grammars (CFG). In [**9**], CFG are used in a different manner. While generating whole sentences with the help of CFG, some types of phrases (figurative phrases, metaphores, similes, oxymorons) are generated using CFG and these are used later on in the actual poem.

2.1.2 Templates

Templates are blocks of text that have gaps in them. When these gaps are filled with appropriate words, a sentence is created. If the filler words are chosen well, the generated sentence will conform with the language's grammar.

The drawback in using templates is that the generated poems will not be very creative, and after a while many poems that are very similar will be generated. If the poem form is not constrained in grammatical structures that it should use, this means that the abilities of a template based poetry generator is very limited compared to a real poet writing in the same genre.

2.1.2.1 Sentence Templates

This kind of templates are used in [**10; 11; 12; 13**]. In this type, templates are actual sentences of which some words are deleted. That is to say, they contain some words that come from the original sentence and the system fills the gaps with the words it chooses.

2.1.2.2 POS-Tag Templates

In this type of template, every element in the template is a POS-Tag. Thus, no words remain from the original material from which the template was crafted. Then, words with appropriate POS-Tags are selected and placed in the template to generate a grammatically correct sentence.

We notice that in [**4**] and [**14**], this method is used in exactly the same way to generate haiku. The authors analyse a corpus of haikus to find the *most common* patterns of POS-Tag

sequences. This is possible because haikus are short and similar in grammar constructions. In haiku, "the wording, not the form variations play the most important role" according to [**14**].

In [**7**], the POS-Tag templates are used as constraints on poem lines. Given a knowledge base containing words with their POS tags, a constraint solver is then used to satisfy these constraints -along with others, relating to meaningfulness and poeticness- with the word set.

Also, the two template approaches may be combined like in the system Pemuisi. [**15**] This actually is equivalent to the first approach but allows for more variety in terms of the words that can be taken out of the original sentences, since in each word's place, the POS tag will be written and the replacing word will be chosen accordingly, without the fear of replacing a noun with an adjective.

2.1.3 n-gram Language Models

n-gram models are very often used to check how well a given sequence of words fit the language of a corpus. In [**16**], there are modules that generate text endlessly. The output from these modules are then processed to create the poem, but the content is always generated by these modules and the grammaticality is supposed to be ensure by this mechanism. In another approach [**17**], the randomly generated texts are ranked using an n-gram model to filter out grammatically erroneous lines.

2.1.4 Recurrent Neural Networks

Recurrent Neural Networks (RNNs) are widely used in generating Chinese poetry [**5; 18; 19**]. Particularly, in [**20**], RNNs are used to approximate the probability of a character at the i^{th} line given the previous characters in the same line and the $i-1$ previous lines. The objective for training in this study is the cross entropy error between the character distribution of the corpus and the predicted distribution.

Seperately, for rhythmical English poetry, the system Hafez [**21**] uses a finite state machine. A path in the machine from a start node to an end note corresponds to a poem (that fits poeticness constraints), though a meaningless one. Then, RNNs are used to select a fluent path, that is, a grammatically correct one.

2.2 Satisfying Meaningfulness

Meaningfulness constraint is the fact that a poem should convey some message, under some interpretation. It should be semantically coherent and not empty. As discussed above, this constraint can be seen as somewhat at odds with the poetic license idea. It is hard to set the threshold of how much ambiguity is allowed. For this reason, Manurung's original constraint states that a poetry generator should generate poems conveying a semantic message *intentionally*. This means that in his system [**3**], the generator tries to convey an exact semantical message, given by the user as input. We will consider other less strict methods, using the assumption "if the selected words of the poem are coherent, then the poem is likely to convey a semantical message, especially with the help of poetic license".

Notice that satisfying Grammaticality does not imply satisfying Meaningfulness. A famous example by Noam Chomsky demonstrates this clearly: "Colorless green ideas sleep furiously." [**6**]. However, this is true for using only the grammar. The other forms of ensuring grammaticality (i.e. template based -especially sentence templates- and statistical language modeling -n-grams and RNNs- based approaches) carry information from the corpus that is used to train models or generate the templates. Therefore, they may be helpful in creating texts that are more meaningful than random. Indeed, the authors of some papers do not include any mechanisms to ensure meaningfulness. [**7; 10; 16**]

2.2.1 Realizing a Target Semantics

In [**3**] and [**8**], the system takes an input from the user. This input is the semantics that the poetry system will try to realize in its poem. In both works, a flat semantic representation is used: the semantics is coded as first order logic predicates (specifically, Prolog syntax is used). While generating the poems, the difference between the target semantics and the semantics of the generated poem up to that point are compared.

In McGonagall[**3**], Manurung describes a way of computing the semantics of a given LTAG tree that has been anchored. It is a complicated process which includes defining features for anchor words and propagating and matching features between grammatical units. Once the semantics that a candidate poem is computed, the semantic difference between this semantics and the target is computed. The difference is used in the design of evolutionary evaluation functions: the systems conveying more of the target semantics takes a greater fitness value. Also, genetic operator are designed that change the candidate poems' structure in a way that will include the unrealized portion of the target semantics and will exclude the semantics that are not part of the target.

In Manurung's chart generation system [**8**], the semantic difference is used to make choices on which sentences will be generated. The initial phrases to be generated are those that realize words in the target semantics. Then, the process continues with the goal of generating the rest of the target semantics.

2.2.2 Using Semantically Coherent Words

This approach is based on the assumption that using semantically coherent, similar or related words will help the poem in having a meaning. There are three main approaches: using datasets or word relations, using data from the web and word embeddings.

2.2.2.1 Using Datasets

In [**11**], Oliveira uses a public lexical resource for Porteguese. It contains triples of form <*word 1, relation_type, word2*>, an example would be "<*vehicle,hypernym,car*>". He then uses these words to fill templates which fit the associated relation types and this contributes to the poem's semantical coherency.

The system in [**9**] uses the WordNet and WordNet-Affect resources to select words for poetry generation that are related in some ways (synonyms, antonyms etc.) and that follow a particular sentimental pattern (also modeled within the scope of the poem). It also uses a resource called Brown Corpus to generate words that are frequently used together.

Gaiku system [**4**] also uses WordNet, but it also uses Word Association Norms which the authors claim is a better resource for discovering novel and surprising relations between words, which is more desirable for the poetry generation task. Word Association Norms first originated in psychology research. It is the words that pop into the subjects' mind after being given a cue word. The dataset contains the cue words and a number of associated words along with statistical measures. The study uses University of South Florida Free Association Norms dataset.

In order to generate association that are novel, they create a graph of words from the word associations and conduct several random walks of short length on the graph. They collect the words they encounter and use these words in the poem. This gives obvious associations as well as more deeply, unexpectedly related words.

2.2.2.2 Using Data from Web

The web has lots of data that can be processed to pull insights regarding the use of language in all kinds of aspects and contexts.

Some poetry generation systems [**10; 15; 14**] simply use the web to find content to investigate and find a topic or sentiment according to which the poem will be written. However, some papers use the web to generate many interesting representations of words and their relationships.

[**14**] uses the number of search results on Google to measure the relatedness or coherence of two words. It also searches the web for each line of the generated poem. The authors argue that if the search yields is none or a few entries then the line is probably nonsensical. They filter out such lines.

Veale [**12**] uses the web in various ways to find out stereotypes, common sense knowledge and relations between entities, feelings and adhectives. He uses techniques like searching for queries, "as X as Y" or "X like Y" to find out similes between nouns X and Y. Also, Google search query completing system is used to find out prejudices by asking "Why do Xs <relation> Ys" to find out about relations. Furthermore, Google n-grams[1] are used for finding out metaphores (for example, looking at n-grams of form "tyrant Adolf Hitler", "racism is a disease" etc.) Also he mines the internet for descriptions like "feeling <feeling> by <noun>" to find out about the emotional connotations of words. Then he defines sets that keep the words coming from each of these different techniques for each word. After this, we can combine these sets to generate new descriptions or relations in terms of what we already know. We then fit these findings in templates to present them in a coherent and grammatically correct fashion.

[1] https://books.google.com/ngrams

2.2.2.3 Extracting Information from Data

Under the fairly reasonable hypothesis that semantically related words are frequently used together and as such, can be detected from the natural language usage. This assumption has proven itself useful.

In particular, in [**13**], a technique to generate a word association graph from raw text is presented. The vertices of the graph are distinct words and log likelihood ratio test is used to determine how related two words are. Then, a threshold is applied to the scores to determine which words will have edges between them.

In more detail, we use the log likelihood ratio test the difference between the probabilities of words occuring together under the assumptions that they are dependent or independent. It first uses the maximum likelihood estimate from the data to estimate how likely the words are to occur independently and together. Then, we use the probabilities of the words occuring independent of each other to estimate the probability that they occur together to get a new probability distribution. Then we test how different these two distributions are and conclude that the words are related if there is a significant difference. We do this for every two words.

Another, more sophisticated method for extracting semantic information from raw text data is word embeddings. Word embeddings are vector representations of words such that the similarity of the vectors reflect the similarity of the words. In Hafez [**21**], after receiving a topic word from the user, a word2vec model is used to select the most similar words to the topic word.

Word embeddings are also useful because they can be used to generate poetry in a **context aware** fashion. That is, instead of choosing coherent words and using them, we choose words according to their coherence with *previously chosen words*, the rest of the poem.

In [**20**], a context vector for all the previously generated lines are calculated from the embeddings of the characters that are used in those lines. Then, for each character, the character's embedding is used to calculate its probability of appearing given this context vector and also the embeddings of all other previously selected characters in the same line.

Also, again in the context of Chinese poetry, the system iPoet [**5**] considers in selected characters' relevance with each other while generating the poem. It follows an approach where characters that are not very coherent with the other characters get replaced by new ones, all using word embeddings of characters. This system also generates embeddings for words in its own way: using Latent Dirichlet Allocations. This returns, for each character and a predefined number of topics, the probabilities that each word is related to the topics. This is then used as an embedding to cluster the characters in terms of topics and use the clusters to generate the actual lines in the poem.

Though not related to word embeddings, another paper employs a technique to generate poems in a context aware fashion. [**17**] Here, as described in 2.1.3, the authors use an n-gram language model to produce grammatically coherent lines. After the first line is generated, using IBM Model 1 and the corpus, every word's unigram probability is computed (given the words in the previous line) and these probabilities are incorporated in the n-gram model, achieving context aware generation.

2.3 Satisfying Poeticness

Poeticness constraint states that poems should conform to some formal and rhythmic constraints. Formal constraints are about the structure of the poem (its stanza and rhyme patterns) whereas the rhythmic constraint expresses the metre of the lines, based on stress, duration or tone of the syllables. It is arguably the most important constraint because while the other constraints are applicable to any natural language generation task, this one is the one that makes a text a poem.

2.3.1 Smart Generation

The most guaranteed way to satisfy this condition is to design the system such that no poem that doesn't conform to this constraint is generated during the process.

Hafez [**21**] uses a finite state acceptor (FSA) such that a path from the inital node to a final node generates a stream of words that satisfy the poetical constraits of sonnets. The FSA machine has nodes for each syllable in the poem (labeled L1S1 for "line 1 syllable 1") and edges are labeled with words. An edge with a word w is present between nodes $LlSs_1$ and $LlSs_2$ is present if the metre allows going from the s_1^{th} syllable to s_2^{th} syllable. Nodes that are at the end of the lines are connected via punctuation marks to the starting nodes of the next line. Lastly, rhyming words to be used at the end of each line are defined and only the edges corresponding to those words are left in the corresponding places in the FSA, deleting all other edges. Under this construction, a path from node $L1S1$ to $L14S10$ defines a sonnet meeting both rhyming and metrical constraints.

In [**17**] a weighted finite state transducer (FST) and EM training method is used to learn stress patterns of words from a corpus of Shakespear sonnets. Then, the authors reverse the FST to generate words complying to a given stress pattern. Then they repeatedly use this machine to generate lines that comply with the user supplied target metre for generating rhythmic poetry in English.

In [**8**], chart parsing method is reversed to generate poems. Only the phrases which have stress patterns such that they can be used in a line are generated, the others are discarded. Thus, the generated lines all comply to the metrical constraints.

In [**7**] and Pemuisi [**15**], both rhyming and rhythmical constraints are implemented in a constrained programming manner and the solutions found by the solver have to be in compliance with the poetical constraints of the chosen poetic form.

2.3.2 Elimination

Another approach to satisfy poetic constraints would be to generate lots of different candidate poems or candidate phrases to be used in poems and then eliminate those that do not fit the constraints.

In [**20**], after learning probabilities for words, authors use a stack decoder which implements tonal pattern and rhyming constraints.

In [**16**] and [**9**], many agents work together to generate a poem. While some are busy with generating words, phrases or poems, others are given the responsibility of evaluating these outputs. These agents send the sketches that don't fit constraints to the trash bin.

2.3.3 Iterative Methods

Another approach is to start with a solution, and then iteratively modify it to reach a solution that fits the constraints as much as possible.

This is the essence of how the McGonagall system [**3**] solves the problem. It employs evolutionary methods to solve the problem of poetry generation, which involves evaluating the fitness of candidate solutions in a generation. By incorporating the poeticness constraints in the fitness function, Manurung expects the last output of the system to obey these constraints.

Also, in [**5**], an iterative method is used. The system reaches a poem and then iteratively replaces characters with others to maximize a utility function. It incorporates the tonal constraints of Chinese poetry in this utility function that it tries to maximize.

2.4 Evaluation Methods

While Manurung's threefold criteria is generally accepted and referenced [**9; 2**], there are some other techniques used by the researchers to establish the quality of the poetry their systems generate.

Many works test their poetry using human judges. Hafez is evaluated using a website by the users with 5 star scoring system. [**22**] Sometimes these scores are compared with authentic human poetry scored by the same people, not knowing which poems were artificially generated. [**14**] Also, sometimes the participant are only asked to seperate human written poetry from machine generated ones.[**4**] This is the so called Turing Test of computer generated poetry.

Another approach is the FACE descriptive model for evaluation of creative systems. In [**2**], Oliveira describes it as

> "To be assessed positively by this model, a creative system must create a concept (C), with several examples (E), include an aesthetic measure (A) for evaluating the concept and its examples, and provide framing information (F) that will explain the context or motivation of the outputs."

The model's creator Colton uses it [**10**] as well as some other researchers. [**9**]

Some works try to incorporate evaluation scores from translation [**20**] and summarization [**5**] domains into poetry generation. Lastly, some scores from NLP research like perplexity [**20**], average cosine similarity and pointwise mutual information are used. [**2**]

2.5 Works on Ottoman Turkish

In the field of computer engineering, the work on Ottoman Turkish is mostly in computer vision and computational linguistics and the research generally aims to help the research on other fields by trying to make it easier to deal with the noisy data and automate mechanistic parts of the work.

In the computer vision side, the work is focused on optical character recognition and document retrieval. [**23; 24; 25; 26**] The word in computational linguistics is about poetry analysis [**27**] and translation/transcription [**28**].

CHAPTER 3

Data

Data in Ottoman Turkish is scarce and most of the time, it is very hard to extract information from. This is because the original data is mostly handwritten Ottoman scripts and only a portion of them has been trancripted in Latin alphabeth and only a small fraction of these has been digitalised.

Most of the data is in image form. Also, the data in Latin alphabeth comes in two forms: the standard Turkish alphabeth and Turkish transcription alphabeth. Thus the data is highly nonstandard.

What makes it even harder to extract information from the data is the nature of Ottoman Turkish. We have no tools for grammatical, lexical or morphological analysis tools for Ottoman because it uses a mixture of Turkish, Arabic and Persian linguistic structures and uses only specific parts of these language so that no existing tool fits with the task.

3.1 Ottoman Text Archive Project

Ottoman Text Archive Project (OTAP) [1] is a research project realised with the joint efforts of researchers from Washington University and Bilkent University. The project defines its mission as:

> "*to create resources that will enable scholars, researchers, decision-makers, and the general public to understand better a critical area of the world and one of the last great multi-ethnic empires.*"

in their website.

Particularly, they digitise the manuscripts from some Ottoman poets, Ottoman novels and other writings. The poems are particularly interesting for this study:

(1) Ghazals of Necâtî
    This part of the dataset includes 650 ghazals which amounts to a total of 4266 couplets consisting of . It includes 13341 distinct words. Only 38% of these words are used more than once, only 19% is used more than twice. This number drops down to 3% for words used more than 15 times.
(2) Ghazals of Mihrî Khatun

[1] http://courses.washington.edu/otap/

Mihrî Khatun's dîvân includes 1350 couplets and 6121 distinct words, 33% of which are used more than once. This number is 8% for more than twice and 2% for more than 15 times.

(3) Divan of Revânî

Dîvân of Revânî is composed of 3868 couplets. It contains 12675 distinct words. 39% of these are used more than once and 11% more than twice. Only 3% are used more than 15 times.

The number of common words found in the 3 poets' poems are as follows:

- 2763 words are used both by **Mihrî** and **Necâtî**
- 2549 words are used both by **Revânî** and **Necâtî**
- 4771 words are used both by **Mihrî** and **Revânî**
- 1992 words are used by **Mihrî** and **Necâtî** and **Revânî**

All of this is in very standard, clean form and in transcription alphabeth. Also, since all three works were composed in the first quarter of the 16^{th} century, they should have a consistent use of language and so the data doesn't suffer from incoherence due to the change of language through time.

3.2 Digital Safahat

The Directorate of Religious Affairs of Turkey has converted *Safahat* (the collected poems of Mehmet Akif Ersoy) in digital format and shared it online[2].

Being written in a much later period, the language in these works are very close to modern Turkish. But the poetry and language use are still those of a master and provide clean data for learning.

This dataset is not written in strictly ghazal form. Therefore, it is not composed of couplets. We have 12016 lines of poetry in this book, which amounts to about 6000 couplets with 21951 distinct words.

34% of the distinct words are used more than once in the data set. This number drops to 20% for more than twice, and to 2% for more than 15 times.

3.3 Dictionaries

In [**28**], an anonumous blog[3] is referenced. There, someone has accumulated 10 different dictionaries of Ottoman Turkish and modern Turkish as database files.

[2] https://safahat.diyanet.gov.tr/

[3] http://ekitapgunlugu.blogspot.com/2013/03/osmanlca-sozluk-veritaban.html

CHAPTER 4

Methodology

In this chapter we will reflect on the ghazal generation problem in light of the literature. We will then try to explain our reasoning in selecting a technique from the literature and discuss its advantages and disadvantages for the ghazal generation task. Afterwards, implementation details will be discussed.

4.1 Ghazals from a Poetry Generation Perspective

As we discussed earlier, poetry can be considered an easier task compared to others because of the issue of poetic license. This *grammatical* freedom is present in the case of ghazal generation too. However, *formal* constraints are very tight and make the job very hard. Rhyme constraints -though not tedious to satisfy- have to be satisfied. There are ways in which metre constraints can be violated but these can not be abused.

On the other hand, ghazals are restricted in their tone and subject. This shrinks the size of our vocabulary and means that a treatise similar to word embeddings will produce results in fine grained detail compared to general purpose embeddings.

There are also commonly used similes, imageries and phrases that may be captured. The only problem is that these require big amounts of data and the data is scarce for this application. We also can not use data from the net or any other contemporary data source because of the language.

On another aspect, we have explained the hardness of evaluation for the works in computational creativity in general. Ghazals make it even harder because the language is not currently used, thus only an expert would be able to make sense out of the produced poem, and being an expert, it is safe to assume that she would be able to differentiate the real poems from the artificially written ones.

Furthermore, there are formal peculiarities that are not addressed in the literature because they are special for ghazals: one example is the usage of pen names. In Ottoman poetry, each poet has a pen name and they must include it in the last couplet of their ghazals. This kind of constraint is not addressed in the literature. Also, as mentioned above, most NLP tools do not work with the ghazal data.

To end the section on a good note, we note that the couplets in ghazals are independent pieces. Each couplet is closed in itself semantically. Thus, the ghazal generation task can actually be reduced to the task of generating couplets complying to a given metrical form and rhyme scheme.

4.2 Contemplations on the Literature

We have seen different methods of ensuring or opting for grammaticality, meaningfulness and poeticness.

Systems that explicitly try to ensure meaningfulness and grammaticality, like those of Manurung [**3; 8**] fail to generate long texts because their search space is very constrained. Even worse, they require us to explicitly define the semantics to be conveyed. While evolutionary approach or chart generation seems to be plausible for ghazal generations, these facts make it seem hard to realize by these methods. Also, these techniques require grammar constructs and general purpose dictionaries, which we can not get our hands on in the case of Ottoman Turkish. Also, they provide no way of ensuring the general tone and subject of the poem will be like that of a ghazal's.

Text template based methods are very limited in their creativity. When combined with hard constraints of ghazals, they would fail to generate interesting results. While POS Tag template approaches are hard to realise because we don't have a good POS Tagger for Ottoman Turkish.

Works that require using datasets about meanings of words or data from the outside world such as the web are out of question because of lack of data in our domain.

RNN based deep learning methods are very promising and are successfully used for English and Chinese poetry generation. However, they are very hard to incorporate with the rhyming and metric constraints of ghazals.

Actually, these constraints are very hard to satisfy and the approach we will take needs to somehow use them to our advantage. We are going to have a hard time if we do not befriend these constraints. As we mentioned earlier, many people find constraints to be the driving force of creativity and good poetry. Constraints can be seen as making our task easier in that *they immensely narrow down our search space*. Then, we can use other techniques to search this narrow space.

While our last paragraph makes it sound like constraint programming approaches are the way to go, as we discussed, they are not applicable in the absence of POS tags. Therefore, we choose to take the finite state machine based methods of Ghazvininejad et al., i.e. the system Hafez [**21; 22**] to be the basis of our system because it lets us use the constraints to meet our goals easier and also employs deep learning techniques which show great promise.

4.3 Method

We decided to base our system architecture on that of the system Hafez [**21; 22**] in trying to generate ghazal couplets. This is because Hafez ensures rhythmic and rhyme constraints and uses an RNN to search for a semantically coherent couplet in this constrained space, just as we need to do for ghazal couplets, thus is clearly addresses all our criteria.

In this section we will give the overview of the Hafez system, then go into the details of the system. We will not explain how Hafez works exactly, but we will describe Binârî, a variant of Hafez, which is designed for ghazal generation.

4.3.1 Overview

Let's first briefly describe the Binârî system:

(1) Binârî uses the data from OTAP[**?**] because it is fairly standardised and of considerable size, compared to the alternatives.
(2) Couplets are semantically closed and independent. Therefore, the task of generating ghazals reduces to the problem of generating couplets which obey a certain rhyme scheme, which reduces to finding rhyme words and generating couplets using those words. Due to lack of data, we can't expect to have learn good word embeddings to use for rhyme word selection. Thus, for now, we take the rhyme constraints as input from the user and try to generate a couplet satisfying the constraints of rhyme, metre, grammaticality and meaningfulness.
(3) In order to generate the couplet, Binârî first prepares a vocabulary from the input file.
(4) It then generates a finite state transducer which has states for each syllable in each line of the poem and state transitions that are labeled with words. If a word gets us from a syllable to another, while complying with the rhythmical constraints, we put an arc labeled with that word between the mentioned syllables' states.
(5) Since we already know the rhyme words i.e. the endings of the lines, we delete all arcs except the rhyming words from the endings of the corresponding lines in our FSA.
(6) Finally, we select a fluent path in the FSA by executing a beam search on an RNN that is trained on OTAP corpus in order to generate a poem.

We now visit each part of the system, explain them in detail and present our proposed changes to make the system compatible with ghazal generation.

4.3.2 Generating the FST

We said that Hafez generates a FST with a state for each syllable in the poem with arcs between them labeled with words that make us go from one syllable to the other. Let's give some examples to clarify what this means.

Imagine we are trying to write a sonnet. We need 14 lines with 10 syllables each and such that each line complies to an alternating pattern of unstressed and stressed syllables. This is called the *iambic pentameter*. Then the first syllable in the poem needs to be unstressed. The second syllable of the first line (denoted *L1S2*) needs to be stressed, and so forth.

Our FSA for a sonnet contains $14 \times 10 = 140$ states labeled $LxSy$ where $1 \leq x \leq 14$ denotes the line number and $1 \leq y \leq 10$ denotes the syllable number. Now, the word *poem* has the stress pattern *stressed, unstressed* and thus can occupy the second and third syllables

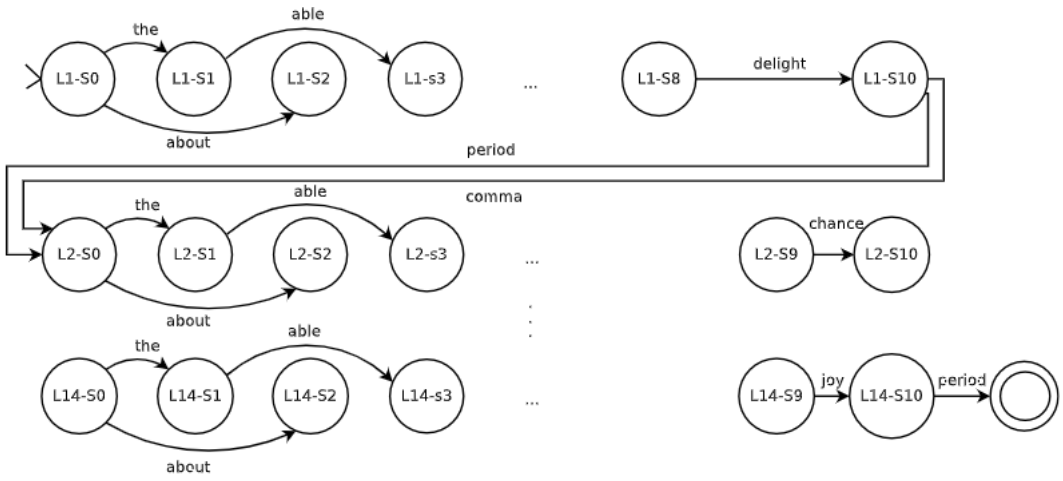

FIGURE 4.1: Figure showing an example FSA from [**21**]. Originally captioned: "An FSA compactly encoding all word sequences that obey formal sonnet constraints, and dictating the right-hand edge of the poem via rhyming, topical words *delight*, *chance*, ... and *joy*"

in a poem in iambic pentameter. Thus, in our FSA, there will be an edge from *L1S2* to *L1S4* which is labeled "poem". Note that the word can occupy many other places without disrupting the rhythmic metre and also it can be present in all the lines, therefore there will be many arcs labeled with the same word.

Continuing our example of sonnets, the ending of an i^{th} line is connected to the beginning of the $(i+1)^{\text{th}}$ line with an edge labeled with a <*endLine*> token.

After constructing the FST, we constrain it to use the rhyme words. Now, for the rhyme word w to be used in the n^{th} line, we remove all edges necessary from the FSA to make w the only word which ends that line. This final touch makes sure that when we follow a path from the state *L1S1* to *L14S10*, if we write down the labels of the arcs we selected, it constitutes a string of words that, though meaningless, comply to all formal constraints of sonnets. Figure 4.1 shows an example FST that is given in the original paper.

In the case of the ghazal, the situation is nearly identical. First of all, as we said, we will only generate a couplet with the FST thus the FST for Binârî is considerably smaller. Instead of stresset and unstressed syllables, we have long and short syllables. In conclusion, FST ensures rhyme and metrical constraints in both cases.

4.3.3 Training the RNN

We now need to select proper words from the FST. We do this by first training a recurrent neural network to get a language model.

Since the Turkish language is agglutinative, variations on a single word generate many different words. This means, using words as the input tokens is not an option with low amounts

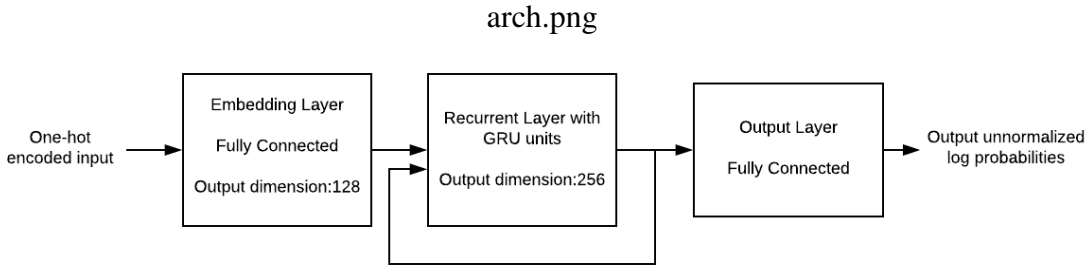

FIGURE 4.2: Neural network architecture of Binârî

of data, because the network can not learn the connections between the many variations of a single word.

In order to remedy this, we used characters and syllables as training tokens to arrive at two different models.

Each model accepts a one-hot encoded input and outputs a probability distribution. Due to lack of data, as we said, we can not learn effective embeddings, therefore, we add a fully connected layer to embed the input into 128 dimensional space, of which output is fed into a recurrent layer. The output of this layer is connected to a fully connected layer which is used to calculate unnormalized log probabilities for the next token.

4.3.3.1 Recurrent layer

In the original paper [**21**], two LSTM layers are used. Since we are limited by the low amount of data we have, Binârî uses one layer with GRU units. Hidden state size is 256.

4.3.4 Decoding the FST with the RNN

Decoding is done via a beam search. Each beam state contains the information on the currently generated text, current hidden state, current FST state and the score of the beam state. A beam size of 5 is used because it severly affects the run time. Moreover, for beam sizes up to 15, we see that all couplets generated are nearly identical. This is mostly because we don't have enough data to train a competent language model. Instead, our model converges to a local minimum which does not make much sense.

4.3.4.1 Beam Search

We define a beam as a 4-tuple: (*generatedText*, *FSTState*, Each beam represents a piece of text written, and a score is associated with the text. The score of a word, given a hidden state h for the RNN, is defined as:

$$\frac{1}{len(word)} \sum_{t \in word} \log p(t|h_t - 1) \tag{4.1}$$

where $word$ is a list of tokens(characters or syllables). For a token t, $p(t|h)$ denotes the probability given by the RNN to the token t and the hidden state h_{t-1}. As for the hidden

states: $h_0 = h$ and $h_t = F(h_{t-1})$ where F is determined by the parameters of the last layer of the neural network.

For each word that has an edge going out from the FST state, we generate a new beam state with the word concatenated to the original beam's text, the new FST state, the new hidden state and the new score, which is found by adding the score of the word given the hidden state h of the beam.

Don't forget that we are choosing the rhyme words in advance. It is not at all probable that the rhyme word will appear at one of the final beam states. This is why we generate the poem **in reverse**. We start with the rhyme words and let the RNN generate the previous words by following paths in the reversed FST.

4.3.5 Implementation Details

4.3.5.1 Finite State Transducer

We implemented the finite state transducer in Python3.

First of all, functions for syllable analysis are created. A function called *get_syllables* accepts a word as string and returns the syllables constituting that word. After this point, it is straightforward to find the rhythmic metre corresponding to the word: syllables ending with a short vowel are short, syllables ending with either a consonant or a long vowel are short and syllables ending with a long vowel followed by a consonant are counted as two syllables, one long and one short, in this order.

In order to find the syllables of words, we used a logically reduced version of the rules defined in the website of Turkish Language Society[1]:

- Each syllable contains only one vowel
- A consonant between two vowels forms a syllable with the vowel following itself
- Any other consonant form a syllable with the vowel preceding itself

A class called *FST* which takes a metre and a filename containing all possible words in its inititalizer is written. This initializer creates the first version of the machine, which only includes metre constraints. Later on, the *constrain* function is called with the rhyme words and information on which line to constrain. This modifies the machine to make the lines end with the given rhyme words. This function can also be used to constrain the machine to include the penname token.

In the end, this machine will be reversed via the *reverse* function before being used in conjunction with the LSTM.

4.3.5.2 RNN Training

As we saw in 3, most of the words in our data is used only once or twice. This means the information that some words are connected will not be given to the RNN and it will most

[1] http://tdk.gov.tr/icerik/yazim-kurallari/hece-yapisi-ve-satir-sonunda-kelimelerin-bolunmesi/

probably not learn anything. In order to remedy this we will change the scope of our tokens and will try different approaches.

(1) **Character level training:** As a baseline, we will try training the RNN with characters as tokens. We are not worried with generating words that actually exist because we are already scoring words from the FSA via the LSTM and selecting the word that has the highest score. However, this approach will probably suffer from coherence between words since LSTM will need to have a very long term memory in order to remember the context from previous words whilst continuing in a character level.
(2) **Syllable level training:** Another way to define subword tokens is to use the syllables themselves. This approach is certainly worth a try because syllables form a natural choice for subwords and we already have written functions to find out the syllables of a given word.

We have used tensorflow[2] with keras[3] to implement the model. For the first fully connected layer, we used the keras Embedding layer utility.

Since our data does not come in a fixed size, we have padded the inputs with zero at the end and have masked the padding tokens during training.

We have tested our system with 500 and 1000 epochs of training.

4.3.6 Second Model

Notice that, given that we don't have much data, it is a very big constraint on the poem that we only select from the vocabulary. We trained the network at the syllable and character levels, thus the network itself is assumed to have learned (or theoretically, there can be a network that has learned) some relationships between the words and their inflections, conjugations etc. This knowledge can not be used if we only choose words from a fixed vocabulary.

As a second model, we let a neural network, trained on the syllable level, generate a string, one syllable at a step. Only thing is, we constrain the syllables it can choose to those which satisfy the metrical constraint. This method is conceptually equivalent to creating the FST with a vocabulary of syllables instead of words. Of course, we need the network to be able to use the space token, so we add " " as a syllable that does not count as a syllable in the metrical metre and is not required to fit any constraints on where it can be used.

Trying this, we actually rely on the model to have learned the language so that it does not produce gibberish words, whereas in the first model, we were sure to generate only words that exist in the language.

[2] https://www.tensorflow.org
[3] https://keras.io

CHAPTER 5

Results

In this chapter, we will look at 4 poems generated by the three models: character level, syllable level and second model.

We thought of scoring the poems according to our criteria and having a numerical measure of performance for the models. However, since all poems satisfy poeticness and none satisfies meaningfulness and grammaticality, since the model can not learn a good language model due to lack of data, it is impossible to score these poems with a precision that would differentiate them. Because of this, we continue with a qualitative discussion.

We present for each model couplets generated with 500 and 1000 epochs with rhyme constraints or no rhyme constraints using the metre "·−−−"x4, using beam size 5. After this standard setup, we share some couplets generated in other settings.

5.1 First Model

5.1.1 Character Level Model

5.1.1.1 Standard Setup Couplets:

	Rhyme Constraints: cān, sūzān	No Rhyme Constraints
500 epochs	ne it ḫūbān-ı ḫıffet ẕevḳı ḫıffet mehliḳā-yı cān ki ṭoz ḫıffet idin ʿışḳumda ʿışḳum ʿışḳunuñ sūzān	ḳu-ı ʿışḳıñ diyen ḫōrşīdden ḫōrşīd ʿışḳumda ki ḫaṭ ḫoşdem-i içsem ʿışḳunuñ ḫorşīd ʿışḳumda
1000 epochs	ne ḫoş-ı faḫr ḫayrān eyleyüp şeydālanursın cān bu ḫaṭ-ı ḫaşm ḳoc ḫūbān ḳoyub-ı vaṣldan sūzān	bu ḫūn-ı ḫışm-ı ḫūbān felekler gülbeşekkerdür ne ḫōş ḫūbān-ı ḫayrān ḫayr-ı maḥmūd ibrām

5.1.1.2 Additional Couplets:

ger ḫo-yı ḫoşdem-i ʿışḳum ẕevḳı ḫıffet ʿışḳumı
kim ḫadin-i ḫaṭṭ-ı ḫōrşīd ʿışḳıñ ʿışḳuma

5.1.1.3 Comments:

We see that the RNN is somehow obsessed with some characters and/or some words and keeps selecting them.

5.1.2 Syllable Level Model

5.1.2.1 Standard Setup Couplets:

	Rhyme Constraints: cān, sūzān	No Rhyme Constraints
500 epochs	şu -ı -ı derc aḳşemseddinüñ ekşitdügüñden cān ne boş ir ʿışḳdur efsāne aḳşemseddinüñ sūzān	bu az baġdāddan ekşitdügüñden ābdāruñdan bu ur güstāẖlıḳ maʿmūresinden bīsaʿādetler
1000 epochs	ye boş illāh aḳşemseddinüñ ekşitdügüñden cān ḳo zehrālūd illa'llāh müşgāsāsıdur sūzān	bu ẖad-ı fenn aḳşemseddinüñ ẖörşīdruẖsārı di boş aġyāra şekkerrīzden ẖörşīdruẖsār

5.1.2.2 Additional Couplets:

mülk-i ıl cüst maḥbūbu'lḳulūbi'lʿārifīn
cān u çıñ muṣḥaf melekruẖsārı şeftālūlerüñ

şu it-i eyleyüb ekşitdügüñden ẖān mısın kāfir
ki-i bülbüllerin allāhdan sūzān mısın kāfir

be ʿışḳum ıl alur ekşitdügüñden ẖān mısın kāfir
ne iç bülbüllerin allāhdan sūzān mısın kāfir

ne -i alnuñ müsāfirẖānedür ekşitdügüñden cān
ne tel altunlu illa'llāh müşgāsāsıdur sūzān

5.1.2.3 Comments:

Here we see the results on training with syllable level data. The model seems to prefer long words whenever it can and it then must choose short words to fit the length of a line. These oddities suggest something went wrong in the training process. However, when we let the models generate text, without being constrained by the FST, we see that these oddities don't persist. Actually, the text generated by the free model seems to be much more convincing, while still very unconvincing.

Generating after the initial string: "<endCouplet>"

	Character Level	Syllable Level
500 epochs	<beginCouplet>baña gelmeye ġamze o meh ṭarḳadurur ṣaʿvaẓı<endLine>gül yine yanmaḳ pelersin zülfüñüñ yerlüce<endLine><endCouplet>	<beginCouplet>ol ṣaḥruñ saña sīm ü lāle bendeñdür kim<endLine>rengīze sisütün üstürenme-i mercān yatur<endLine><endCouplet>
1000 epochs	<beginCouplet>yazalum bu ḳıplaġız mı ider ü cānı<endLine>senüñ olanı sökeṣdürse loḳmān ehli<endLine><endCouplet>	<beginCouplet>eyle yüz girlüme urmış begüm miskīn tekellüf hīç<endLine>sürer sürer ṣu yolına yine iken dāyim beyigi<endLine><endCouplet>

These are the texts generated by the neural network when it is not constrained in any way. We see that these models output something closer to natural language. This is why we had the idea of letting a syllable level RNN generate text one syllable at a time and restraining it to choose syllables that defy the rhytmhmic metre.

5.2 Second Model

5.2.1 Standard Setup Couplets:

	Rhyme Constraints: cān, sūzān	No Rhyme Constraints
500 epochs	ki gūyā mülkiçün bāleb-i pürçünkim ḳalis-i cān ṣafaḳdur ḫāne-i devlet serāser mest-i sūzān	bu ṭūbā çünki ṣol kim gül çerāzın bezmümüzdüb ḳış ṣa-yı miḫnet anuñ kim anmazın ancaḳ kemer dirler
1000 epochs	eger çün seyr hey hey bezmhānuñçün benümçün cān yiyā ṭāvūsdur gülgūn ṣabā olmış berā sūzān	ʿaceb mül teşnedir kāfir muḳılmaḳ etdi gerlārī ʿacebdür bir ḳadem ṣanmañ anuñ efsānedür dirler

5.2.2 Additional Couplets:

ne çün her bir içün ümmīd-i çīn pes nenüñ ger cān
se ter-i zer ʿazānuñdan anuñ ḥācet degül sūzān

geyer bir sāḳıyā sen gülbenāhey itse it hey cān
ʿalem ḳurdān gülistānuñda ṣāhum ṣābihā sūzān

5.2.3 Comments:

Using this technique, the model doesn't repeat any character or syllable or word relentlessly. It can also now use its knowledge at the subword level and generate novel words, some of which may be nonsense.

We see that with this much data, the neural network can not perform well under poetic constraints. While we know that the training process works and that it is learning something, it can not perform well when constrained. A way to see that the training is working is this: Due to the way syllables are defined on a word, in Turkish, a word can not include a syllable that end with a consonant followed by a syllable that starts with a vowel. Indeed, if the neural network selected syllables with this pattern, it would do errors in the metre, but it does not because it has learned this property of the input sequences. Therefore, our main problem is insufficient data.

CHAPTER 6

Conclusion and Future Work

In this project, we could implement a procedure by which a good language model can be used to generate good poems. However, the data in this field is not nearly enough for the required neural network performance.

Therefore, the first thing to do is find or prepare more data. This may include sophisticated methods like OCR or may mean finding a way to learn from modern Turkish data and to transfer that knowledge to tha ghazal domain.

If we had no problems regarding data, or if we solve these problems, there are a lot of things to add to this model. Some are

(1) Use word embeddings instead of a vanilla layer.
(2) Make the model more complex
(3) Make use of NLP tools to find better subword tokens which will provide better results in grammaticality and meaningfulness. Morphological analysis would be the natural direction to try first.

Putting the data problem aside, we need to solve the problem that all beams returning from a search are nearly identical. This may or may not be related to the poor performance by the neural network.

References

[1] S. Colton, G. A. Wiggins, *et al.*, "Computational creativity: The final frontier?," in *Ecai*, vol. 12, pp. 21–26, Montpelier, 2012.

[2] H. G. Oliveira, "A survey on intelligent poetry generation: Languages, features, techniques, reutilisation and evaluation," in *Proceedings of the 10th International Conference on Natural Language Generation*, pp. 11–20, 2017.

[3] H. Manurung, "An evolutionary algorithm approach to poetry generation," 2004.

[4] Y. Netzer, D. Gabay, Y. Goldberg, and M. Elhadad, "Gaiku: Generating haiku with word associations norms," in *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pp. 32–39, Association for Computational Linguistics, 2009.

[5] R. Yan, H. Jiang, M. Lapata, S.-D. Lin, X. Lv, and X. Li, "I, poet: automatic chinese poetry composition through a generative summarization framework under constrained optimization," in *Twenty-Third International Joint Conference on Artificial Intelligence*, 2013.

[6] P. Gervás, "Computational modelling of poetry generation," in *Artificial Intelligence and Poetry Symposium, AISB Convention*, 2013.

[7] J. Toivanen, M. Järvisalo, H. Toivonen, *et al.*, "Harnessing constraint programming for poetry composition," in *The Fourth International Conference on Computational Creativity*, The University of Sydney, 2013.

[8] B. C. L. Tobing and R. Manurung, "A chart generation system for topical metrical poetry.," in *ICCC*, pp. 308–314, 2015.

[9] J. Misztal and B. Indurkhya, "Poetry generation system with an emotional personality.," in *ICCC*, pp. 72–81, 2014.

[10] S. Colton, J. Goodwin, and T. Veale, "Full-face poetry generation.," in *ICCC*, pp. 95–102, 2012.

[11] H. G. Oliveira, "Poetryme: a versatile platform for poetry generation," *Computational Creativity, Concept Invention, and General Intelligence*, vol. 1, p. 21, 2012.

[12] T. Veale, "Less rhyme, more reason: Knowledge-based poetry generation with feeling, insight and wit.," in *ICCC*, pp. 152–159, 2013.

[13] J. Toivanen, H. Toivonen, A. Valitutti, O. Gross, *et al.*, "Corpus-based generation of content and form in poetry," in *Proceedings of the third international conference on computational creativity*, University College Dublin, 2012.

[14] R. Rzepka and K. Araki, "Haiku generator that reads blogs and illustrates them with sounds and images," in *Twenty-Fourth International Joint Conference on Artificial Intelligence*, 2015.

[15] F. Rashel and R. Manurung, "Pemuisi: a constraint satisfaction-based generator of topical indonesian poetry.," in *ICCC*, pp. 82–90, 2014.

[16] P. Gervás, "Tightening the constraints on form and content for an existing computer poet," in *The AISB15's 2nd International Symposium on Computational Creativity (CC2015)*, p. 1, 2015.

[17] E. Greene, T. Bodrumlu, and K. Knight, "Automatic analysis of rhythmic poetry with applications to generation and translation," in *Proceedings of the 2010 conference on empirical methods in natural language processing*, pp. 524–533, 2010.

[18] Z. Wang, W. He, H. Wu, H. Wu, W. Li, H. Wang, and E. Chen, "Chinese poetry generation with planning based neural network," *arXiv preprint arXiv:1610.09889*, 2016.

[19] R. Yan, C.-T. Li, X. Hu, and M. Zhang, "Chinese couplet generation with neural network structures," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, (Berlin, Germany), pp. 2347–2357, Association for Computational Linguistics, Aug. 2016.

[20] X. Zhang and M. Lapata, "Chinese poetry generation with recurrent neural networks," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 670–680, 2014.

[21] M. Ghazvininejad, X. Shi, Y. Choi, and K. Knight, "Generating topical poetry," in *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1183–1191, 2016.

[22] M. Ghazvininejad, X. Shi, J. Priyadarshi, and K. Knight, "Hafez: an interactive poetry generation system," in *Proceedings of ACL 2017, System Demonstrations*, pp. 43–48, 2017.

[23] A. Onat, F. Yildiz, and M. Gündüz, "Ottoman script recognition using hidden markov model," *IEEE Transaction on Engineering Computing Technology*, vol. 14, pp. 71–73, 2006.

[24] E. Ataer and P. Duygulu, "Retrieval of ottoman documents," in *Proceedings of the 8th ACM international workshop on Multimedia information retrieval*, pp. 155–162, ACM, 2006.

[25] I. Z. Yalniz, I. S. Altingovde, U. Güdükbay, and Ö. Ulusoy, "Ottoman archives explorer: A retrieval system for digital ottoman archives," *Journal on Computing and Cultural Heritage (JOCCH)*, vol. 2, no. 3, p. 8, 2009.

[26] E. Can, P. Duygulu, F. Can, and M. Kalpakli, "Redif extraction in handwritten ottoman literary texts," pp. 1941–1944, 08 2010.

[27] F. Can, E. Can, P. D. Sahin, and M. Kalpakli, "Automatic categorization of ottoman poems," *Glottotheory*, vol. 4, no. 2, pp. 40–57, 2013.

[28] J. Korkut, "Morphology and lexicon-based machine translation of ottoman turkish to modern turkish,"