Machine Learning & Data Mining
Miniproject 3

Caltech CS/CNS/EE 155
March 11^{th}, 2018

---

1 Introduction

- **Group members**
  Yukai Liu, Botao Hu and Jian Xu

- **Team name**
  aaaalbert

- **Division of labour**
  Yukai Liu was in charge of RNN.
  Botao Hu was in charge of data preprocessing, Naive HMM and HMM poem generation.
  Jian Xu was in charge of HMM additional goals and visualization.

2 Pre-processing (Part 3)

- **Tokenization**

  Tokenization is a process where we split the whole data set into different units such as words or phrases, and give each unit a unique index to distinguish them. We used `Keras.preprocess.text` package to do the tokenization. The built-in methods in this package is able to split poems into words, and map words to its number of occurrence/index. We do the tokenization based on the following rules.

  - **Punctuation not included.** If we treat punctuations as tokens, we might encounter some totally meaningless sentence segments. We think it is better if we add punctuation manually after the whole poem is generated.
  - **Hyphenated word is a single word.** In writing sonnets, Shakespeare actually create some new word by hyphenating two words together. This way of word-creation is highly praised by the English literature field. If we treat hyphenated words separately, we will lose the uniqueness and creativeness of sonnets.
  - **The apostrophe is maintained in tokenization.** There are several usages of apostrophe in sonnets. Sometimes poet use this sign for omission of syllables (e.g. *o'er-snowed, short for over-snowed, line 8, Sonnet 5, Shakespeare*); in other occasions it is used in passive cases (e.g. *summer's day, line 1, Sonnet 18, Shakespeare*). If we simply treat them as their original form, we might make grammar mistake or syllable mismatch in generating our new poems.

- **Sequence**

  Initially we tried to treat each poem as a sequence, which is more likely to maintain the logic and sentiment of the poet. However, when doing cross validation (train the unsupervised model on 90% of the poems, and predict the probability of the rest of poems), because a poem is a long sequence and the trained matrices are naturally sparse, we often encountered underflow in probability prediction. Even though we used strong regularization, we got $\mathbb{P}(\text{validation sequence}) \sim 10^{-300}$, which is too small for us to do hyper-parameter validation. Furthermore, it is hard to satisfy line-wise requirements, such as syllable, rhyme and meter (we don't even know how long the generated sequence should be).

We finally used each line as a sequence to train our model. In order to take care of rhymes, we **reversed** the words in a line before we plug them into the model. For specific implementations, please see Poem Generating and Advanced Techniques part.

- **Training data set & Specialized dictionaries**

  We incorporated the poems of Shakespeare and Spenser as one large training data set. We generated two specialized dictionaries to map syllables and rhymes of each word that appears in the data set. Please see Advanced Techniques for details.

3 Hidden Markov Models

- **Naive HMM (Part 4)**

  - **Naive unsupervised learning:** Naive HMM actually tries to maximize the probability of the training dataset with the given parameters - the transition matrix and the emission matrix. In the unsupervised setting, we are given a training set of N training examples containing only the sequence $\mathbf{x}$'s:

    $$S = \{x\}_{i=1}^{N}$$

    and the maximum likelihood problem is thus:

    $$argmax \prod_{i=1}^{N} P(\mathbf{x}_i) = argmax \prod_{i=1}^{N} \sum_{\mathbf{y}} P(\mathbf{x}_i, \mathbf{y})$$

    . If we knew the marginal distributions $P(y_i^j = a, \mathbf{x}_i)$ and $P(y_i^j = b, y_i^{j-1} = a, \mathbf{x}_i)$ that we got from Forward-Backward from HW6, we could use the training data to estimate the parameters of our HMM model as:

    $$P(y_i^j = b | y_i^{j-1} = a) = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M_i} P(y_i^j = b, y_i^{j-1} = a, \mathbf{x}_i)}{\sum_{i=1}^{N} \sum_{j=1}^{M_i} P(y_i^{j-1} = a, \mathbf{x}_i)}$$

    $$P(x^j = w | y^j = a) = \frac{\sum_{i=1}^{N} \sum_{j=1}^{M_i} \mathbf{1}_{x_i^j = w} P(y_i^j = a, \mathbf{x}_i)}{\sum_{i=1}^{N} \sum_{j=1}^{M_i} P(y_i^j = a, \mathbf{x}_i)}$$

    Similar with HW6, we used EM algorithm to solve the optimization problem.

  - **Regularized unsupervised learning:** Furthermore, in order to avoid the sparsity of the transition matrix A and emission matrix O, we add a regularization term on both A and O. When we are calculating A, we initialize A with a $n \times n$ matrix with all entries equal to λ_A, and initialize O with a $n \times d$ matrix with all entries equal to λ_O. Here, n is the number of the hidden states, and d is the number of the emitted tokens that we pre-defined. Therefore, we could use the training data to estimate the parameters of our HMM model as:

    $$P(y_i^j = b | y_i^{j-1} = a) = \frac{\lambda_A + \sum_{i=1}^{N} \sum_{j=1}^{M_i} P(y_i^j = b, y_i^{j-1} = a, \mathbf{x}_i)}{n^2 \lambda_A + \sum_{i=1}^{N} \sum_{j=1}^{M_i} P(y_i^{j-1} = a, \mathbf{x}_i)}$$

$$P(x^j = w|y^j = a) = \frac{\lambda_O + \sum_{i=1}^{N}\sum_{j=1}^{M_i} \mathbf{1}_{x_i^j = w} P(y_i^j = a, \mathbf{x}_i)}{nd\lambda_O + \sum_{i=1}^{N}\sum_{j=1}^{M_i} P(y_i^j = a, \mathbf{x}_i)}$$

We also used EM algorithm to find the solution of the optimization problem.

- **Cross validation:** We used 10-fold cross validation during our training process finding the optimal number of hidden states. We split the dataset (consisting of both Shakespeare and Spenser) into 10 subsets, and used 9 of them as training data and 1 of them as validation data. From the training curve, when the number of hidden states is 6, the cross validation probability gets the maximum.

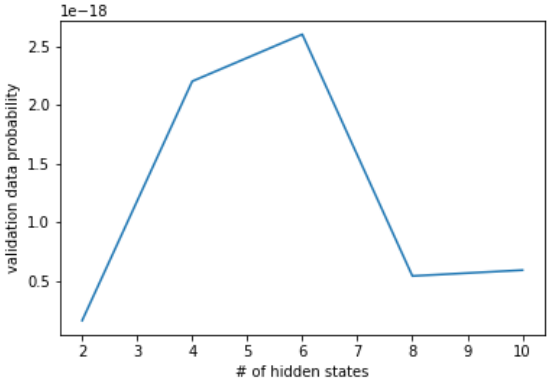


Figure 1: Training curve

- **Poem Generating (Part 5)**

  - **Naive version: poem-as-sequence HMM**

    As mentioned in the Pre-processing part, we first used each poem as a data sequence. In this approach, we generate poem with a fixed length, which is the average length of words of poems in the data set. The generated poem is shown below on the left, with rough sentence segmentation and very simple punctuations (comma at the end of odd sentences, period at the end of even sentences). As we can see from the generated text, the poem is full of grammar mistakes, and makes no sense at all. So we change the idea into using lines as sequences.

Table 1: **Poem Generation (left: naive, right: advanced)**

are a long the jacks now remembered,	*Most wrongfully pains mow religious plot,*
to life my time evil all shun whom thou.	*Wherein entrap my heart are not grow sum!*
dost brass love being unused since limbecks,	*Lightening clear till never-resting blot,*
self are fears my pass offices birth those.	*Exceeding saints outcast interchange some.*
forsake the the at the upon imperfect hate's,	*Heretofore 'tis wait of would wastes old eyes,*
my ow'st kind glad thou what for my sacred.	*Corruption graced of witness appeal.*
my wert give to dear doth of eyes sweetest,	*Setting cloudy belong plain doth the wise,*
tattered thorns a me your question eye curse.	*Flattery fair unworthy why ne'er feel?*
and counterfeit not me thee my thou,	*Columbines doth o, but breasts ere so hand,*
therefore children's to hate will by not.	*Devouring night meant 'Sorrows heat frost themes'.*
my will youth be as by mow cheater me,	*Injury despite dropping tempered stand,*
a thou former thy dear, my so not heir.	*Benefit greater for did lilies beams.*
than self with art inhabit prey so he eyes,	*External but in fulfil purer make,*
doth stol'n me victors resort more calls.	*Becoming do antique thee abstain take.*

– **Advanced version: line-as-sequence HMM**

We used reversed line as sequences to train the HMM model. In sequence generation, we start with the last word of the line. At first we randomly choose a hidden state to start with, and sample an emission word according to the observation matrix. The sampled word should satisfy (1) it has a clear rhyme in the rhyme dictionary (see Advanced Techniques below); (2) it has a clear number of syllables in the syllable dictionary (also see Advanced Tenchniques below); (3) there are other words in the dictionary which has the same rhyme as the sampled word (so we can generate another corresponding line). If not, we sample the word again until we find some "good" starting point. We use a `rhyme` variable to record the rhyme of this line. There is also a `remain_syllable` variable to record how many syllables left in this line (it starts with 10 - number of syllables in the start word).

Then we do a simple loop to generate words in the middle of the line. After sampling hidden state from the last hidden state, we sample words which has a clear number of syllables in the syllable dictionary. We update `remain_syllable`, and we emit that word.

When we find `remain_syllable` less than or equal to 3, we immediately exit the above loop. This time we sample the last word of this line. The last word should satisfy (1) it has a clear number of syllables in the syllable dictionary; (2) its number of syllables is equal to `remain_syllable`. In Shakespeare's Sonnets, a word may have a different syllable number when it is at the end of a line, so we should also take this factor into consideration. We then emit this word, reverse the word order, and return the result line.

Now we come to the second line which has the same rhyme as the previous one. The whole process is the same, except that at start, we only sample words which has the same rhyme as `rhyme` variable. This time, we make a new list only consisting of rhyme words (except the word used in the last line), and normalize their probability so that we can only sample from this list.

In this way, we get two lines of the same rhyme, and the numbers of syllables of these lines are 10.

Using the same strategy, we generate other 6 line pairs. After that, we reorder these lines to match the rhyme requirement, and return the whole poem. Also, we manually add some punctuations according to the grammar and sentiment of this poem. The generated poem by the advanced approach is shown above on the right. As we can see, the poem has better grammar and richer sentiments. It is more readable since we specially designed syllables and rhymes. The only drawback is that the lines are not consistent in logic and emotions, and this is because we separate lines apart for training.

The table above shows our best attempt of poem generation with 6 hidden layers, and the table below is a comparison between other numbers of hidden states. We couldn't tell much difference about these results; but we do see that poems in table 2 tend to be less logical and meaningful than the poem above.

Table 2: **Advanced Poem Generation with Various Numbers of Hidden States**

2 hidden states	4 hidden states	8 hidden states
warriors so torture heart knife unless	*summer upon disabled rest son*	*idea plead heart a breathe veil sharpened*
invocate i one for form cries thee thy	*forget'st water should thou o and didst dove*	*issueless score weighs prayers quickly look for*
needing wasteful liken your like tombed dress	*dying universe trifles their two run*	*nothing disdaineth inquire imprisoned*
impression can kingdoms very that eye	*another's these that soonest crying of*	*deceased eyes a seeing why spring more*
heaven women's the deprave themselves praise	*directed lets mansion eclipse reckoning*	*cuckoo expressed and love this resty bends*
wiped unkindness to it ever it prayers	*relenting sorrow sing this eke dead things*	*legacy mask disabled awards*
wandering then with vile deeds increase days	*vouchsafe corruption riches gaol slandering*	*celestial that yield thou write daily spends*
imprisoned mortality doth affairs	*barrenly whose time thou new humble kings*	*counterfeit or care the to art we wards*
sorrow echoes form up disabled	*ornament the pointing spent easy curse*	*dignified the monsters side direct gate*
depend betwixt and decay wantonly	*surety-like i thy nerves muse state pride parts*	*entertain perceiv'st in and if although*
dissolved compare such sighs quite enfeebled	*aggravate powerful tongues purging disperse*	*applying because my smell make mind hate*
supposed thine false thy in the happy	*unjustly proves scarcely kindling it heart's*	*offenders kindle true mind's wondrous so*
triumphant conscience of which well debtor	*wilfully what when day face fury shake*	*well-tuned some sweet for that sessions hide*
untimely itself level nerves after	*violet any to doing scope make*	*evermore beggared my zealous abide*

[Some insight on poem generation] One may doubt about this generation method: why don't we just create states with regard to syllable and rhyme factors, plug them as part of the data into the unsupervised training process, and let the model freely "express" its poems? The reason is that if we consider these factors before training, we will use (far) more states to train this model. The more parameters we have, the more we will fit the noise, and the more likely we will have overfitting issues. In this case, the HMM model is not as free as one may think; we actually confine this model to have limited idea: it loses variety in generation. On the other hand, the semi-supervised learning method we use is more difficult and tricky in designing generating process, but is less prone to overfit the data set.

- **Advanced Techniques (Part 7)**
  - **Rhyme:** The NLTK library `http://www.nltk.org/_modules/nltk/corpus/reader/cmudict.html` provides pronunciation information for almost all the English words (127069 entries). The `cmudict` dictionary uses 39 phonemes to describe the syllabus, and they are shown in the Table 2 below. One example of syllabus of a long word is

`'judgement': ['JH', 'AH1', 'JH', 'M', 'AH0', 'N', 'T'].`

Following the nature of rhymes, we start at the end of the syllabus list and traverse back, and fetch all the phonemes we meet as rhyme until we found a vowel phoneme, which has a number as the last character. For example, the rhyme for word `judgement` is `['AH0', 'N', 'T']`. There is also a list of words (in the Shakespeare and Spenser data set) having the same rhyme:

```
["ornament", "evident", "judgement", "constant", "moment", "tyrant", "silent", "triumphant",
"argument", "excellent", "absent", "defendant", "present", "instant", "judgment", "servant",
"sequent", "vacant", "monument", "extant", "patent", "fragrant", "abundant", "different",
"truant", "pleasant", "patient", "frequent", "accident"].
```

In this way we can construct a rhyme dictionary mapping rhymes to lists of words. In sequence generation, we can just look up the dictionary to find words with proper rhymes.

Table 3: Phonemes Correspondence

Phonemes	Word Example	Phonemes	Word Example	Phonemes	Word Example
AA	odd	AE	at	OW	oat
AH	hut	AO	ought	P	pee
AW	cow	AY	hide	S	sea
B	be	CH	cheese	T	tea
D	dee	DH	thee	UH	hood
EH	Ed	ER	hurt	V	vee
EY	ate	F	fee	Y	yield
G	green	HH	he	ZH	seizure
IH	it	IY	eat	OY	toy
JH	gee	K	key	R	read
L	lee	M	me	SH	she
N	knee	NG	ping	TH	theta
UW	two	W	we	Z	zee

- **Meter:** Due to the difficulties in finding reliable source of word stresses, we skipped the stress factor and focus mainly on syllable count. Now that we have a dictionary of syllabus, it is easy to find the number of syllables inside a word: counting the occurrence of vowels. In the example above, word `judgement` has two vowels, so it has two syllables. We apply this method to words that only appear in the Spenser data set. For Shakespeare, thanks to the provided dictionary, we can have all the information of Shakespeare's wording. In this way, we also created a dictionary that records the number of syllables for all the words (note that some Shakespeare's words may have different syllables at the end of a line, and these words are saved under another special key).
- **Additional texts:** We incorporated Spenser's poems as additional training set. We treated them equally in pre-processing and training. We randomly mixed the order of poet of these two poems, in order to avoid extreme validation cases (i.e., train the model on Shakespeare's poems and ask it to predict on Spenser's poems).

4 Recurrent Neural Networks

- **Naive character-based RNN (Part 6)**

  - **Model Introduction:** We used Pytorch, which is more friendly for dynamic graphs than TensorFlow or other packages, to build and train our RNN models. In terms of the specific model we used, instead of LSTM, we used GRU because GRU has similar structure as LSTM but has fewer parameters and is easier to train, so it is more suitable than LSTM for this project. The basic principle of GRU is as follows:

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
n_t &= \tanh(W_h x_t + U_h(r_t \circ h_{t-1}) + b_n) \\
h_t &= (1 - z_t) \circ h_{t-1} + z_t \circ n_t
\end{aligned}
$$

    Variables:

    * x_t: input
    * h_t: hidden state / output
    * z_t: update gate
    * r_t: reset gate
    * n_t: combined information
    * W, U and b: parameter matrices and vector

    There are two gates in GRU: a reset gate and an update gate. The reset gate determines how to combine the new input with the previous memory, and the update gate defines to what extend we will forget previous information.

  - **Training:**

    * As for training data, basically we need to feed every character including punctuations into our model. We first delete unnecessary empty lines and spaces, and then treat every "\n" as a start of a new sequence. Sequence length is set to be 70. For example, seq[i] = '\n', input = seq[i:i+seq_length], target = seq[i+1:seq_length+1]. Training data consists of 90% of the entire data, which has around 3000 sequences with length 70. Validation data includes the rest 10%.
    * As for network structure, we make each character a one-hot vector, and directly feed it into GRU units. Then the output hidden states are passed through a fully connected layer with output size equal to the number of possible characters, each of which represents the probability of picking that character as the next one.We use the negative log likelihood loss as our training target.
    * As for hyper parameters, we mainly care about the number of layers in GRU units, the number of hidden states in each layer, the dropout rate in GRU units and the learning rate.

---

* **Actually in this part we are only asked to look at the training loss and train the model until convergence, so basically we will definitely overfit training set in this process. We will discuss bias-variance trade-off later in the advanced techniques part.** If we just want to overfit training data, any network structure that is large enough can complete this task. In this part, we choose the structure of GRU units to be 2 layers with 512 hidden units in each layer. (Actually if we want to overfit training data, any structure is able to do that.)

Please refer to our code for more details.

– **Poem Generation:** Poem generation comes naturally in RNN. If we don't have a seed, we just start our poem with '\n', feed it into our network, sample the next character from the output distribution, then take this character and hidden states and feed them into GRUs again. If we have a seed, we first send seed characters into networks one by one to initialize hidden states, then we sample the following characters as usual.

With the given seed, using different temperatures, we generate poems as follows (color coding stands for sentences coming from the same original sonnet):

T = 0.25

shall i compare thee to a summer's day?
thou art more lovely and more temperate:
rough winds do shake the darling buds of may,
and summer's lease hath all too short a date:
sometime too hot the eye of heaven shines,
and often is his gold complexion dimmed,
and every fair from fair sometime declines,
by chance, or nature's changing course untrimmed:
but thy eternal summer shall not fade,
nor lose possession of that fair thou ow'st,
nor shall death brag thou wand'rest in his shade,
when in eternal lines to time thou grow'st,
so long as men can breathe or eyes can see,
so long lives this, and this gives life to thee

T = 0.75

shall i compare thee to a summer's day?
thou art more lovely and more temperate:
rough winds do shake the darling buds of may,
and summer's lease hath all too short a date:
sometime too hot the ortaining skill appear,
be my mending on thy fresh respect,
me thou my oblation of her pleasure,
she may detain, but not still keep her treasure!
her audit (though delayed) answered must be,
and her quietus is to render thee,
in others' works thou dost but mend the style
and arts with thourard in my heart compare
with sun and moon, with earth and sea's rich greece,
through hold my life in their dead-doing might

T = 1.5

shall i compare thee to a summer's day?
that neither will for better be a choty,
if will, then she at will may well but will.
that my not to nonouries you had annure,
they had not skill enouress indighined
thou grace when all my best of spring in black,
and each to recompented with swift mesion spench,
she knows, hy fill with thy mother ase
this it no ratters now to hopes,
my love is such shorty, where vain away:
let those tears are tenants to the very.
that censtanct marbless my duty spring:
how heavy if i unseen, though all alie.
.rut him that travels in the joy of me.

We can find that because we are overfitting training data, our model has the ability to memorize all original sonnets. This phenomenon becomes more obvious when T is small. Actually temperature controls our confidence towards the model. When T is small, softmax is very sensitive to small value changes, hence it becomes max function, our model tends to output the character with the highest probability. When T is large, value changes don't have a big influence on softmax output, then the output distribution tends to be uniform. From this point of view, smaller T can produce more reasonable words and sentences, but here when our model just memorizes the sonnets, it will output the exact original sonnet given the first sentence from that sonnet. Larger T makes less reasonable words and sentences, and some of the words it produces are even not real words.

For example, the sonnet generated from T = 0.25 is exact the sonnet 18 with the first sentence. The sonnet generated from T = 0.75 consists of sentences from several different sonnets. The sonnet generated from T = 1.5 does not have the same sentence from any sonnet except for the first sentence.

Generally speaking, this model can capture the sentence structure and sonnet structure. For

example, in the second sonnet, except for some sentences from existing sonnets, we can still find many connections and local grammar structures that are very common in Shakespeare's sonnets. From another perspective, in the third sonnet, even though some of the words do not make sense, it learns when to end a sentence and some useful phrases.

- **Compare RNN with HMM:** Compared with HMM, first of all, because it is a much more complex model in terms of the number of parameters, it is able to capture more details in the sonnet like word-to-word transitions and meters, but on the other hand, when overfitting, it can also just memorize all poems and all structures. Another obvious difference is that because it is a character-based model, sometimes it can't generate complete words.

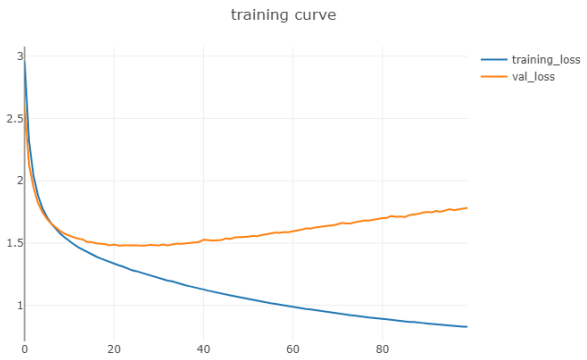
  As for running time, since it has much more parameters than HMM does, it also takes much longer to train. When running on CPU, this model trains very slowly, and the speed becomes acceptable when we start to use GPU. As for the amount of training data required, RNN also needs much more training data than HMM does. One evidence is that this model is easily overfitting training data. As we can see from the training curve below (in Advanced Techniques part), validation error decreases only in the first several steps, and in the whole rest of the training process, when training error still goes down, validation error goes up.

- **Advanced Techniques (Part 7)**

  - **Additional texts:** Because of RNN's hunger for data, we also combine Spensers sonnets in our training.
  - **Using more metrics:** In this part, we will discuss the training process and bias-variance trade-off of RNN in more details. As described above, we divide corpus into training set and validation set. In this part, we did a grid search on the parameters we've mentioned above, the final model structure we choose is: *layers = 3, hidden unites = 256, dropout = 3, learning rate = 0.001*. We train this model for 100 epochs, draw the training curve, and use the model with the best validation error to generate poems. Here we try some different temperatures, table 4 shows its training curve and the poem generated from this model using T = 0.5.

    Here because we are not just memorizing original sonnets, we find these sonnets less complete. Some of the words are not real words, and some connections and punctuations do not make sense. To be honest it is not a good model if we just evaluate the poem it generates, since the amount of data we have is too small compared with the complexity of the model we used. We can also find this fact from its training curve. Validation error only goes down in first several epochs, then it goes up. On the contrary, training error keep going down and it seems it still doesn't converge after 100 epochs. But the advantage of this model is that we can make sure we are generating poems from the essential patterns inside these poems, but not just memorizing some fixed sentences. And because of this we are able to use smaller temperature to increase the completeness of our poems.
  - **Word-based RNN:** Besides char-based RNN, we've also tried word-based RNN. We use the same data preprocessing technique as HMM, and treat each "\n" as a start of a new sequence, choose sequence length to be 15. In this model we still treat each unique word as a one-hot vector. The main difference with char-based model is that there are more than 3000 unique words, so that the

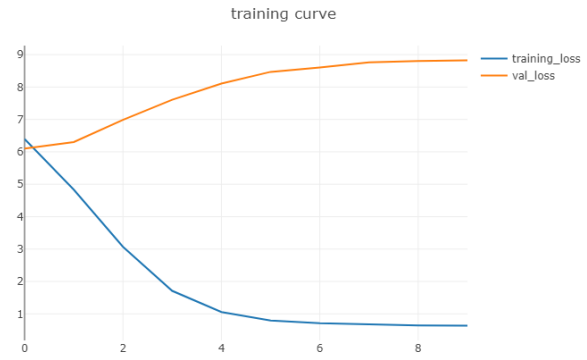
Table 4: training curve and poem of char_based model



shall i compare thee to a summer's day?
then do i not that beauty better to thee,
that when i to the recieuring all their store:
when i sigh the world thee i sake my heart,
and the sun ever for thy self are still to entrate,
but hath the world than thou will, and the fire
that the pleasure of live is thou hast her sad,
or have still me which still the world be beauty:
and their beauty beauty still doth thee,
and the baser the world come to me her.
if it the straight through a steen find and deep,
though her heart doth the therefore in thee to thee,
and then be the world in thing thee better,
the straight with shadow is the dired,

input and output vector size will be more than 3000. Since we already know that RNN can easily overfit training data, here we just care about the model with the smallest validation error. After grid search, the final model structure we choose is: *layers = 1, hidden unites = 512, dropout = 0.0, learning rate = 0.01*. We train this model for 10 epochs, draw the training curve, and use the model with the best validation error to generate poems. Here we try some different temperatures, table 5 shows its training curve and the poem generated from this model using T = 0.75.

Table 5: training curve and poem of word_based model



shall i compare thee to summer's day
tired no offend gain by may form
for limit when suffers pilgrimage die
coral nothing the favourites of papers
have name eyes at me
then in thy beauty grows time
for fear and sickle's dost profound themselves recounting thing
and world before with love's tincture and up away
powerful him thy frank shall sad
shall loathsome am you are
for to slavery and to just my self ill
but not make the world still woe for my didst with prey
and debarred the sufferance slumbers beauty which on of general
though in the world's so or cry

From the training curve above we can find that compared with char-based model, this model

is much easier to overfit training data. To be specific, validation loss starts to go up even only after one epoch. We think one of the most important reasons is that the huge input and output size gives this model too many parameters. Meanwhile there are lots of words that only appear once or twice which makes the prediction not that meaningful. We can also see from the generated poem that even though word-based model can gaurantee all words are real words, but their connections and sentence structures are defnietely unreasonable. If we use the overfitted word-based model to generate poems, we can get similar results as char-based models, which is omitted here.

- **Word-embedding RNN:** Besides tokenizing every word, we used an off-the-shelf unsupervised learning feature extraction package GloVe to do word embeddings (https://nlp.stanford.edu/projects/glove/). As it describes, "*Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.*" Considering the size of our dataset is relatively small, word embeddings are trained on the entire corpus of Shakespearen text, from http://shakespeare.mit.edu/. Then we feed the embedded vectors into GRU units as usual. After grid search, the final model structure we choose is: *embedding vector size = 200, layers = 2, hidden unites = 1024, dropout = 0.0, learning rate = 0.001*.

  In our experiment, there is not so much difference between tokenized word-based model and word-embedding model. One possible reason is that since these models are too complicated and need too many data, they both overfit training data seriously.

Because of time-limit, our experiment is by no means thorough, and one possible problem is that we don't have a deep understanding of how to use GloVe to extract features, if we have opportunities to continue this project in the future, we may try to get better feature representations and combine char-based model and word-feature-based model.

5 Visualization and Interpretation (Part 8)

- **Transition matrix and emission matrix:** HMM learns the poem pattern by finding the hidden states transition properties and the word emission probability at each state. By maximizing the marginal probability of the dataset, we could find the transition matrix A and the emission matrix O.

  From the visualization plots of the transition matrix A and emission matrix O, we can see some entries with much higher values than others as expected sparsity. However, because of the regularization term, instead of equal to zero, those low value entries are equal to a regularization number. This could give the machine poem writer more randomness and help avoid generating monotonous poems that correspond to the minima of the model.

- **Wordclouds:** By using the emission matrix O, we can find the wordclouds corresponding to each hidden state.

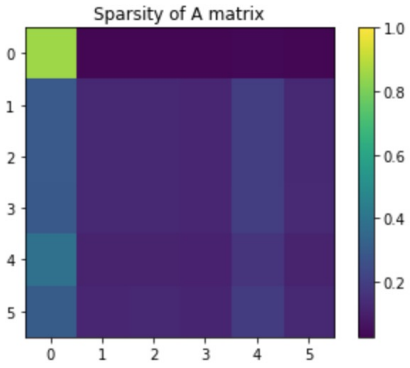


Figure 2: Transition matrix A

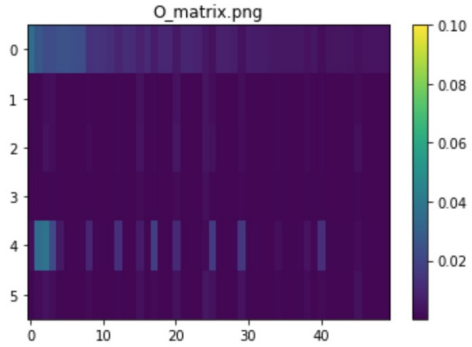


Figure 3: Emission matrix O

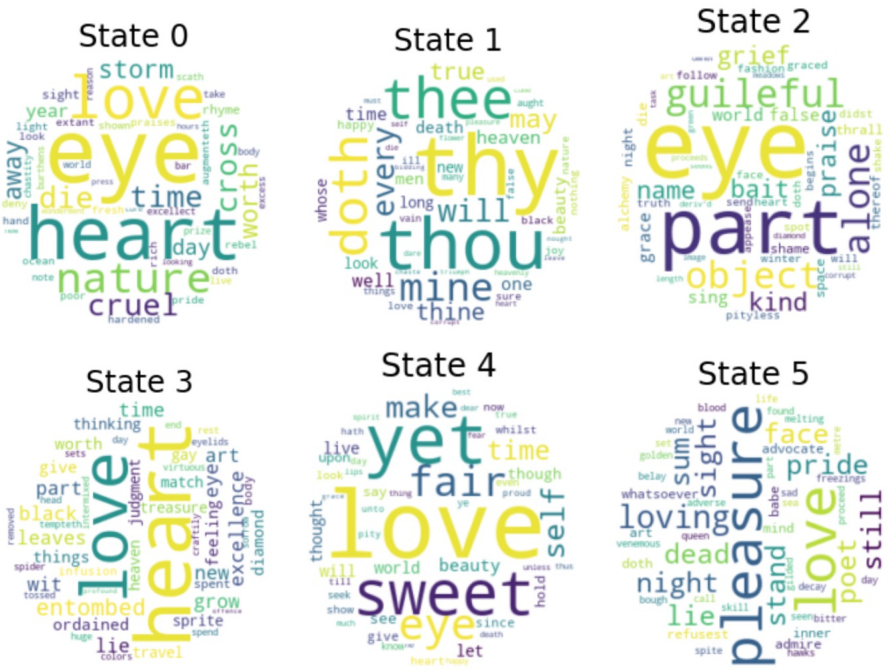


Figure 4: Wordclouds of each state

We also take 10 popular words in each hidden state and make a table. From the table, we can see that the word ”love” almost appears at every state. This does not surprise us since the sonnets of Shakespeare are mostly about love. It is quite obvious that state 1 denotes the pronoun since the top words are ”thee”, ”thou”, ”thy” ... This may indicate that most of the sonnets are about expressing love to the lover and our method does dig out this property of the sonnets. For other states, the properties may not be as obvious as state 1, but we can still find some inner structures. For state 0

Table 6: 10 popular words for each state

State	Popular words									
0	eye	heart	love	nature	time	cruel	storm	die	away	cross
1	thee	thou	thy	mine	will	doth	every	thine	true	mine
2	eye	part	guileful	alone	object	name	kind	praise	bait	grace
3	love	heart	part	excellence	time	thinking	things	lie	wit	sprite
4	love	yet	sweet	fair	eye	self	make	let	live	time
5	pleasure	love	loving	sight	night	lie	pride	face	poet	still

and 5, we think they correspond to the nouns related to love, such as heart, eye, nature, which are objective nouns, or pleasure, pride, which are abstract nouns. For state 4 and 2, they may correspond to the adjectives related to love, such as sweet, fair which are positive, or alone and guileful, which are negative.

- **Transition diagram:** From the transition diagram, we can see that there is a "hot" state (at the right corner) that receives the most arrows. This corresponds to the sparsity of the transition matrix A even if we have added the regularization term. From the wordcoulds, we think the "hot" state corresponds to state 1 because of a lot of "doth", "thou" ... This also verifies the performance of our model since the "hot" state corresponds to the state with the clearest meaning.

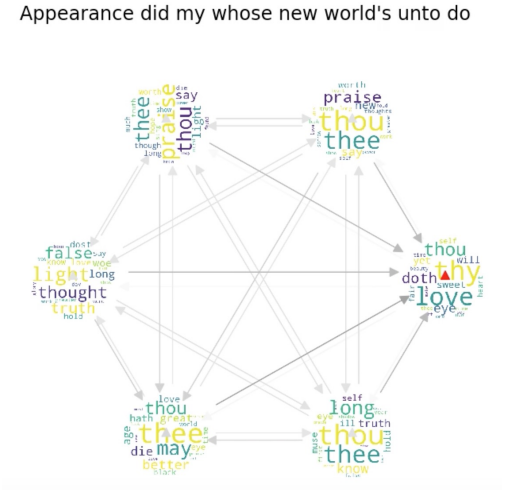


Figure 5: Transition graph