Conversion from Prose to Poetry

GROUP NO. 10

Vineet Jain (14EC35020)
Rajarshi Saha (14EC35030)
Atul Bansal (14EC35028)
Shashwat Mishra (14EC35018)
Akshit Mantri (13EE35009)

Project Description

The problem in hand is to convert a given Sanskrit prose content to poetry format.

Factors that characterize a poetry:

- Presence of Rhyme: Repetition of similar sounding syllables which makes the poem auditorily aesthetic.
- Organisation of the content into verses, as contrasted to paragraphs in a prose.
- Meter: Basic rhythmic structure of a verse or lines in a verse (e.g. Iambic Pentameter)
- Literary devices: Ample presence of literary devices like Imagery, Metaphor, Euphemism, Personification, etc.
- Poetic License: Refers to alteration of the conventions of grammar or language, or rewording of pre-existing text.

Objectives:

Our model should be able to map given sentences in Sanskrit prose to poetry after training. The model should be able to handle the following issues:

1) **One-to-Many mapping**: The same idea can be put forth in several ways by different poets.
2) **Poetic perception is highly subjective**: Because an idea can be versed in various ways, it is often difficult to judge outright whether one poem is better than the other as such a decision often involves a good deal of human intervention. Our model should be able to handle this.
3) **Word Sense Disambiguation**

Experiments and Procedure:

Recurrent Neural Networks (RNNs):

- RNNs are neural networks, specialized for processing a sequence of values x (1) ,..., x (τ) .
- Most networks can process sequences of variable length. It does so by sharing parameters across different parts of a model

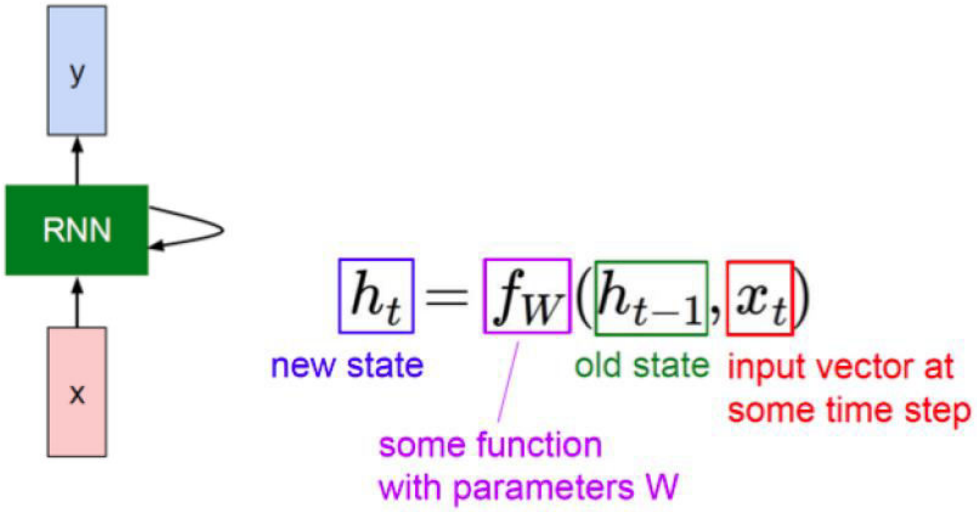

Fig. A sample RNN state update model

LSTM :

Normal RNN has the tendency to forget the past data over a long period of time. LSTM architecture, which is a slightly modified version on simple RNN, takes this into account using forget, input and output gates. This becomes quite helpful for our case as the prose character can have a very high dependency on character coming quite early in poetry.

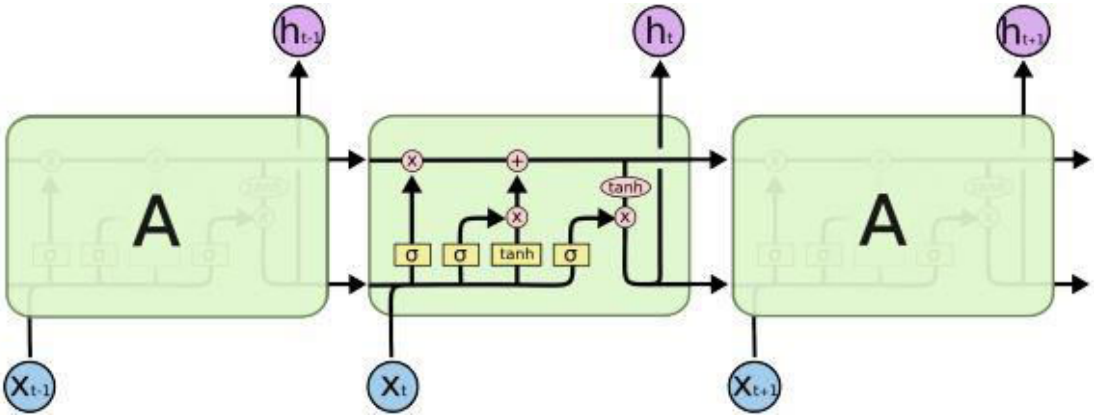

Fig. LSTM model

Our initial model:

Encoder-Decoder

- Input sequence X = (x (1) ,..., x (nx))
- Output sequence Y = (y (1) ,..., y (ny))
- Encoder (reader/input) RNN: Emits the context C, a vector summarizing the input sequence, usually as a simple function of its final hidden state
- Decoder (writer/output) RNN: Is conditioned on the context C to generate the output sequence.
- Note: The lengths nx and ny can vary from each other

Features to be included in our model

- Use an encoder-decoder sequence to sequence architecture with the prose text as the input and the poetry text as the desired output
- Use unigram character embeddings for feeding the text to the network
- This is considered as the baseline model and we will be modifying it to improve performance

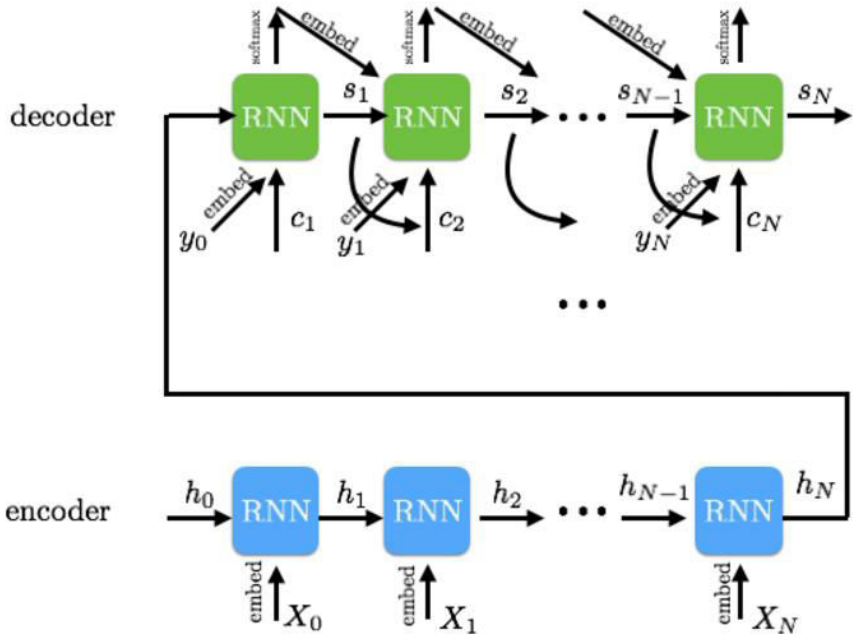

Fig. Encoder Decoder Model

Input Data : One Hot Encoding

- Our model uses unigram characters as inputs and outputs.
- To achieve this purpose we have used one-hot encoding scheme where each character is represented using a 54x1 in which only one element is 1 and others are 0. Thus, providing unique representation for all.
- 54 characters include 26 a-z, 26 A-Z, space( ) and apostrophe(').

  Examples:

a = 0
b = 1
c = 2

a → [1 0 0 0 0 0 ...]
b → [0 1 0 0 0 0 ...]
c → [0 0 1 0 0 0 ...]

Processed Input Data : Character Embedding

The one-hot encoded character is represented by a very sparse vector which unnecessarily increases computational complexity and contains little information. We have therefore used embedding for each character which converts the sparse one-hot vector to a dense vector of shorter length (say 30)

$$y = f(w.x+b)$$

y: Character Embedding output (this will be the input to the Encoder-Decoder)

x: One-hot Encoded Vector

w,b: Function parameters (learnable)

Cost Function used is Cross Entropy

$$D(y',y) = -\sum_j y_j * \ln y_j$$

Results:

Epoch 0
(at start of training)

```
saH vaDyamAna su BfSam BujAByAm pari ghya tO aprakampyO nara vyAGrO rOdraH prasTAtum
EcCata  92
sa vaDyamAna suBfSam BujAByAm parigfhya tO aprakaMpyO naravyAGrO rOdraH prasTAtum EcCata
89
ODe
NTXuNTTXRJiiimmQmQQTDoTTXJGBbNSJPuaaannxGLLLLLJLLJGLLLLLJYLLLLLJYYDLLJXGEEtXiiiimmmQWQDDqD
FFFFmFbUU 103


tasya aBiprAyam ajYAya rAmaH lakSmanam abravIt ayam rAkSasaH anena paTa alam vahatu tAvat
tu  93
tasya aBiprAyam ajYAya rAmo lakzmaRam abravIt vahatu ayam alam tAvat paTAnena tu rAkzasaH
90
OD$ 3
```

Epoch 1

```
t0 Kadg0 kSipram udyamya kfzRa sarp0 iva udyat0 tUrRam ApatataH tasya tadA prahAratAm
balAt  92
t0 Kaqg0 kzipram udyaMya kfzRa sarp0 iva udyat0 tUrRam ApetatuH tasya tadA prahAratAm
balAt  92
tataH samaram samarA samarA samarA samarA samarA samarA samarA samarA samarA samarA
samaram $ 93


saH vaDyamAna su BfSam BujAByAm pari ghya t0 aprakampy0 nara vyAGr0 r0draH prasTAtum
EcCata  92
sa vaDyamAna suBfSam BujAByAm parigfhya t0 aprakaMpy0 naravyAGr0 r0draH prasTAtum EcCata
89
tataH samaram samarA samarA samarA samarA samarA samarA samarA samarA samarA samarA
samaram $ 93
```

Epoch 16

```
t0 Kadg0 kSipram udyamya kfzRa sarp0 iva udyat0 tUrRam ApatataH tasya tadA prahAratAm
balAt  92
t0 Kaqg0 kzipram udyaMya kfzRa sarp0 iva udyat0 tUrRam ApetatuH tasya tadA prahAratAm
balAt  92
tataH samare tataH samare tataH samare tataH samare tataH samarTam anujagmur anupratimam $
90


saH vaDyamAna su BfSam BujAByAm pari ghya t0 aprakampy0 nara vyAGr0 r0draH prasTAtum
EcCata  92
sa vaDyamAna suBfSam BujAByAm parigfhya t0 aprakaMpy0 naravyAGr0 r0draH prasTAtum EcCata
89
tataH samare tataH samare tataH samare tataH samare tataH samarTam anujagmur anupratimam $
90
```

Epoch 24

```
tasya aBiprAyam ajYAya rAmaH lakSmanam abravIt ayam rAkSasaH anena paTa alam vahatu tAvat
tu  93
tasya aBiprAyam ajYAya rAmo lakzmaRam abravIt vahatu ayam alam tAvat paTAnena tu rAkzasaH
90
tataH samare tataH samare tataH samare tataH samare tataH samare tataH samare tataH $ 85


rAkSasaH yaTA icCati taTA vahatu niSA caraH yena yAti ayam eva hi naH panTA  76
yaTA ca icCati sOmitre taTA vahatu rAkzasaH ayam eva hi naH panTA yena yAti niSAcaraH  86
sa tatra viSvAmitram tataH samare tataH samare tataH samare tataH samarTam arhasi $ 83
```

Discussions:

- Frequent characters are learnt quite early as seen in the output producing s,a,t and space quite early onwards
- Space placement is learnt early on with average occurrence after 5-6 characters
- Although the network seem to converge the results are not satisfiable
- Words formed have no meaning generally

- To improve the results, we tried using Adaptor Grammar.

Adaptor Grammar:

Adaptor grammars are a flexible, powerful formalism for defining nonparametric, unsupervised models of grammar productions. The statistical independence assumption used in PCFG's is too strong for modelling the natural language, and thus Adaptor Grammar is used to break that independency.

Reason to choose it:

- PCFG assumes that rewriting operations are independent given the non-terminal.
- This context-free assumption is too strong for modelling natural language.
- Adaptor Grammar transforms a PCFG's distribution over trees Gc rooted at nonterminal c into a richer distribution Hc over the trees headed by a nonterminal c - called grammaton
- These tree-distributions are formed by using a generalization of Dirichlet process

Algorithm 1 Generative Process

1: For nonterminals $c \in \boldsymbol{N}$, draw rule probabilities $\boldsymbol{\theta}_c \sim \text{Dir}(\boldsymbol{\alpha}_c)$ for PCFG $\boldsymbol{\mathcal{G}}$.
2: **for** adapted nonterminal c in $c_1, \ldots, c_{|\boldsymbol{M}|}$ **do**
3: Draw grammaton $H_c \sim \text{PYGEM}(a_c, b_c, G_c)$ according to Equation 1, where G_c is defined by the PCFG rules $\boldsymbol{R}$.
4: For $i \in \{1, \ldots, D\}$, generate a phrase-structure tree $t_{S,i}$ using the PCFG rules $\boldsymbol{R}(e)$ at non-adapted nonterminal e and the grammatons H_c at adapted nonterminals c.
5: The yields of trees $t_1, \ldots, t_D$ are observations $x_1, \ldots, x_D$.