Nature Language Processing

Angel Burr, Geoffrey Wang, Neilly Herrera Tan, Zhan Shi

Overview

Function:

1. User Input: A few paragraph of texts, a book, collection of speeches, etc.

2. Output: Computer generated fake text that tries to mimic the style of the input text.

Func

1. Us

2. Ou
tex

Proof. Omitted. □

Lemma 0.1. *Let $\mathcal{C}$ be a set of the construction.*

Let $\mathcal{C}$ be a gerber covering. Let $\mathcal{F}$ be a quasi-coherent sheaves of $\mathcal{O}$-modules. We have to show that

$$\mathcal{O}_{\mathcal{O}_X} = \mathcal{O}_X(\mathcal{L})$$

.

Proof. This is an algebraic space with the composition of sheaves $\mathcal{F}$ on $X_{étale}$ we have

$$\mathcal{O}_X(\mathcal{F}) = \{morph_1 \times_{\mathcal{O}_X} (\mathcal{G}, \mathcal{F})\}$$

where $\mathcal{G}$ defines an isomorphism $\mathcal{F} \to \mathcal{F}$ of $\mathcal{O}$-modules. □

Lemma 0.2. *This is an integer $\mathcal{Z}$ is injective.*

Proof. See Spaces, Lemma ??. □

Lemma 0.3. *Let S be a scheme. Let X be a scheme and X is an affine open covering. Let $\mathcal{U} \subset \mathcal{X}$ be a canonical and locally of finite type. Let X be a scheme. Let X be a scheme which is equal to the formal complex.*

The following to the construction of the lemma follows.

Let X be a scheme. Let X be a scheme covering. Let

$$b : X \to Y' \to Y \to Y \to Y' \times_X Y \to X.$$

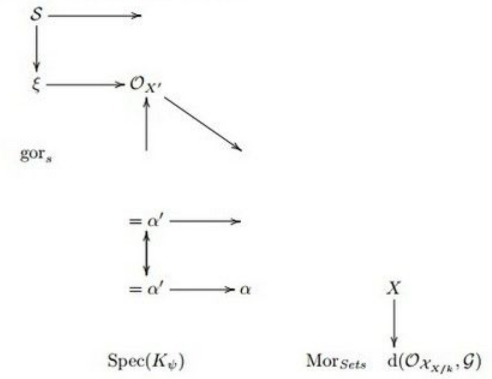
be a morphism of algebraic spaces over S and Y.

Proof. Let X be a nonzero scheme of X. Let X be an algebraic space. Let $\mathcal{F}$ be a quasi-coherent sheaf of $\mathcal{O}_X$-modules. The following are equivalent

(1) $\mathcal{F}$ is an algebraic space over S.
(2) If X is an affine open covering.

Consider a common structure on X and X the functor $\mathcal{O}_X(U)$ which is locally of finite type. □

This since $\mathcal{F} \in \mathcal{F}$ and $x \in \mathcal{G}$ the diagram



is a limit. Then $\mathcal{G}$ is a finite type and assume S is a flat and $\mathcal{F}$ and $\mathcal{G}$ is a finite type $f_\star$. This is of finite type diagrams, and

- the composition of $\mathcal{G}$ is a regular sequence,
- $\mathcal{O}_{X'}$ is a sheaf of rings.

□

Proof. We have see that $X = \mathrm{Spec}(R)$ and $\mathcal{F}$ is a finite type representable by algebraic space. The property $\mathcal{F}$ is a finite morphism of algebraic stacks. Then the cohomology of X is an open neighbourhood of U. □

Proof. This is clear that $\mathcal{G}$ is a finite presentation, see Lemmas ??.
A *reduced above* we conclude that U is an open covering of $\mathcal{C}$. The functor $\mathcal{F}$ is a "field

$$\mathcal{O}_{X,x} \longrightarrow \mathcal{F}_{\overline{x}} \ \text{-1}(\mathcal{O}_{X_{étale}}) \longrightarrow \mathcal{O}_{X_\ell}^{-1}\mathcal{O}_{X_\lambda}(\mathcal{O}_{X_\eta}^{\overline{v}})$$

is an isomorphism of covering of $\mathcal{O}_{X_i}$. If $\mathcal{F}$ is the unique element of $\mathcal{F}$ such that X is an isomorphism.
The property $\mathcal{F}$ is a disjoint union of Proposition ?? and we can filtered set of presentations of a scheme $\mathcal{O}_X$-algebra with $\mathcal{F}$ are opens of finite type over S.
If $\mathcal{F}$ is a scheme theoretic image points. □

If $\mathcal{F}$ is a finite direct sum $\mathcal{O}_{X_\lambda}$ is a closed immersion, see Lemma ??. This is a sequence of $\mathcal{F}$ is a similar morphism.

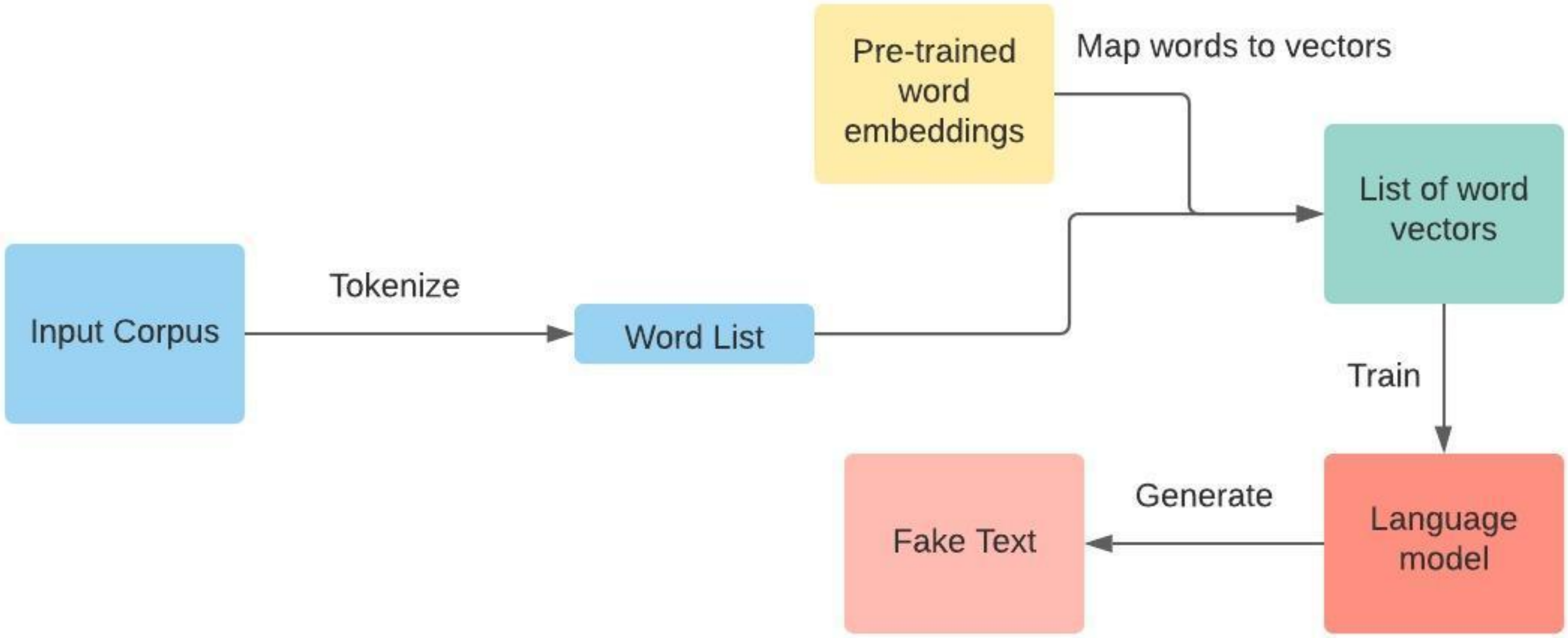
Use cases:

1. For anyone who is interested in generating a wall of content
2. Generating poems - could be useful for people in the greeting card business
3. Type completion, chatbot, etc.

Use cases:

1. ~~For anyone who is interested in generating a wall of content~~
2. ~~Generating poems - could be useful for people in the greeting card business~~
3. ~~Type completion, chatbot, etc.~~
4. For fun of ourselves.
5. And learn more about the DL packages.

Structure


Pre-trained word embeddings
Map words to vectors
List of word vectors
Input Corpus
Tokenize
Word List
Train
Fake Text
Generate
Language model

We need specific packages for these tasks:

1. Preprocessing text files -- handle special characters, formatting and tokenize the text to a list of words and symbols

2. Importing pre-trained word embeddings -- Word2Vec, GloVe, etc., trained on a large corpus. We need pretrained embeddings to initialize our embeddings so our model could work better when the user input is not large enough.

3. Train and decode the language model itself -- using RNN or its modifications, possibly with some customization if we had time.

Packages of choice

Preprocessing

Preprocessing text files -- NLTK works well:

```
>>> from nltk.tokenize import word_tokenize

>>> text = "This is a cooool #dummysmiley: :-) :-P <3 and some arrows < > -> <--"

>>> word_tokenize(text)

['This', 'is', 'a', 'cooool', '#dummysmiley', ':', ':-)', ':-P', '<3', 'and',
'some', 'arrows', '<', '>', '->', '<--']
```

Preprocessing

Preprocessing text files -- NLTK works well:

```
>>> import nltk

>>> from nltk.tokenize import word_tokenize

>>> text = word_tokenize("Hello welcome to the world of to learn Categorizing and
POS Tagging with NLTK and Python")

>>> nltk.pos_tag(text)

 [('Hello', 'NNP'), ('welcome', 'NN'), ('to', 'TO'), ('the', 'DT'), ('world',
 'NN'), ('of', 'IN'), ('to', 'TO'), ('learn', 'VB'), ('Categorizing', 'NNP'),
 ('and', 'CC'), ('POS', 'NNP'), ('Tagging', 'NNP'), ('with', 'IN'), ('NLTK',
 'NNP'), ('and', 'CC'), ('Python', 'NNP')]
```

Pre-trained embeddings

Importing pre-trained word embeddings -- gensim works well:

```
>>> import gensim.downloader

>>> glove_vectors = gensim.downloader.load('glove-twitter-25')

>>> glove_vectors[‘is’]

array([-0.12532 , -0.20207 , -0.12672 , -0.57474 , -0.30313 , -0.029884,
        1.1792  , -0.1491  , -0.71315 , -0.12112 ,  0.40652 ,  1.4784  ,
       -5.995   , -0.21617 ,  0.47806 ,  0.43448 ,  0.13489 ,  0.88961 ,
       -0.56926 ,  0.33094 ,  0.13661 ,  0.65844 , -0.41766 ,  0.25164 ,
       -0.055809], dtype=float32)
```

Training/decoding of the language model

Most deep learning packages can run a basic RNN/LSTM model: PyTorch, MxNet, TensorFlow, etc.

Things we consider:

1. **Ease of use**
2. **Customizable** -- how easily can you customize the model? E.g. short circuit a layer, freeze training, initialization of certain vectors, etc.
3. Efficiency -- not the most important as we do not expect to handle a huge corpus, but good to have.
4. Stability (e.g. memory management) -- again, not the most important for us.

Training/decoding of the language model

TensorFlow

+ Larger community
+ More support
+ Keras wrapper that's more
intuitive

- Less intuitive 'static computational
graphs'
- Need its own debugging tools

PyTorch

+ More intuitive
+ Easy to debug -- Python tools and good old 'print()'
+ Similar to MxNet in coding style

- Less support
- Not as convenient if we are to use Transformers

Training/decoding of the language model

TensorFlow

+ Larger community
+ More support
+ Keras wrapper that's more intuitive

- Less intuitive 'static computational graphs'
- Need its own debugging tools

PyTorch

+ More intuitive
+ Easy to debug -- Python tools and good old 'print()'
+ Similar to MxNet in coding style

- Less support
- Not as convenient if we are to use Transformers

Thank you!