AMYA CODERs

606202001

REPHRASE DETECTION

Instruction Manual

Author:
Atul Sahay

June 16, 2020

Contents

1 Project's Directory Structure

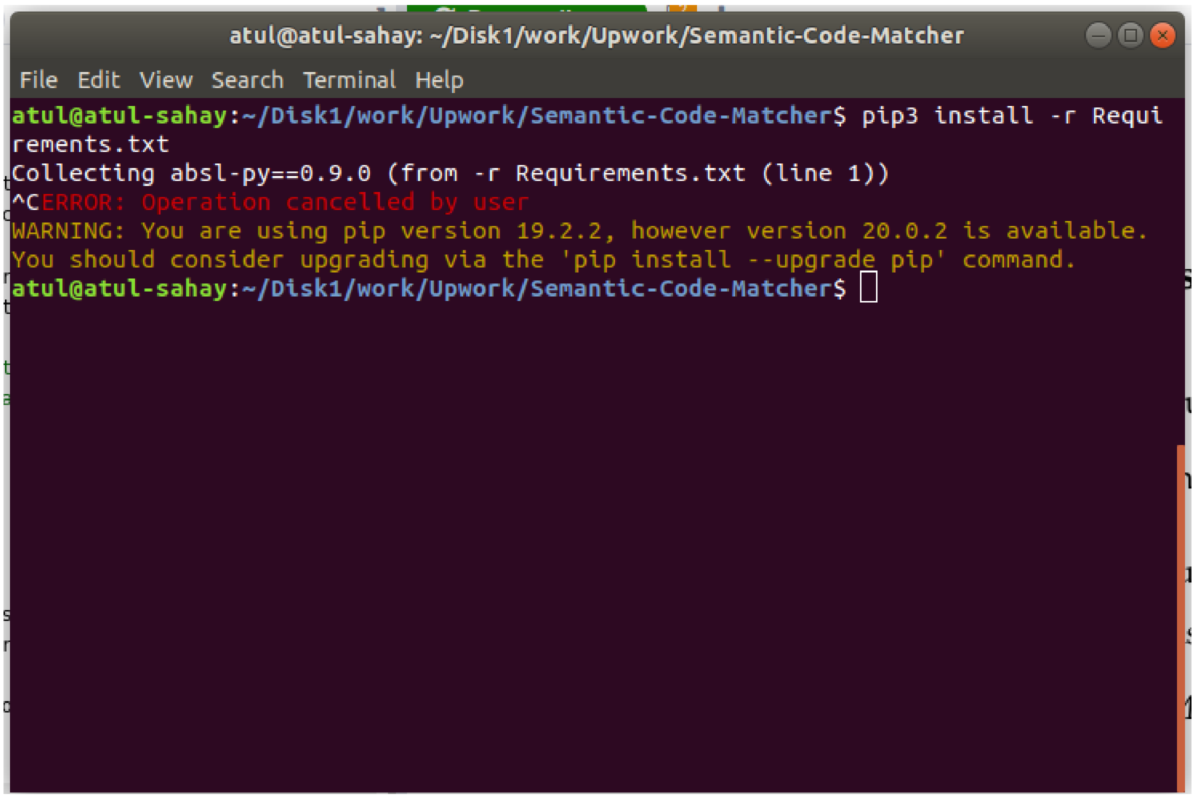
```
Explainable Latent Structures Using Attention
├── Model
│   ├── basic.py
│   ├── _init_.py
│   └── treelstm.py
├── pretrained
│   ├── model.pkl
│   └── vocab.pkl
├── snli
│   ├── Data
│   │   ├── train.pkl
│   │   ├── valid.pkl
│   │   └── test.pkl
│   ├── SaveModel
│   ├── utils
│   │   ├── dataset.py
│   │   └── _init_.py
│   ├── Vocab
│   │   └── vocab.pkl
│   ├── build_vocab.py
│   └── model.py
├── utils
│   ├── glove.py
│   └── vocab.py
├── dump_dataset.py
├── evaluate.py
├── train.py
├── README.md
├── Requirements.txt
└── Rephrase.ipynb
```

2 Requirements

2.1 How to install required modules for the project?

- Search for the **Requirements.txt** file in the project repository.
- Open the terminal and write (be sure you are in the project directory): See Fig: 1

user: Explainable-latent-structures$: **pip3 install -r Requirements.txt**


atul@atul-sahay: ~/Disk1/work/Upwork/Semantic-Code-Matcher
File Edit View Search Terminal Help

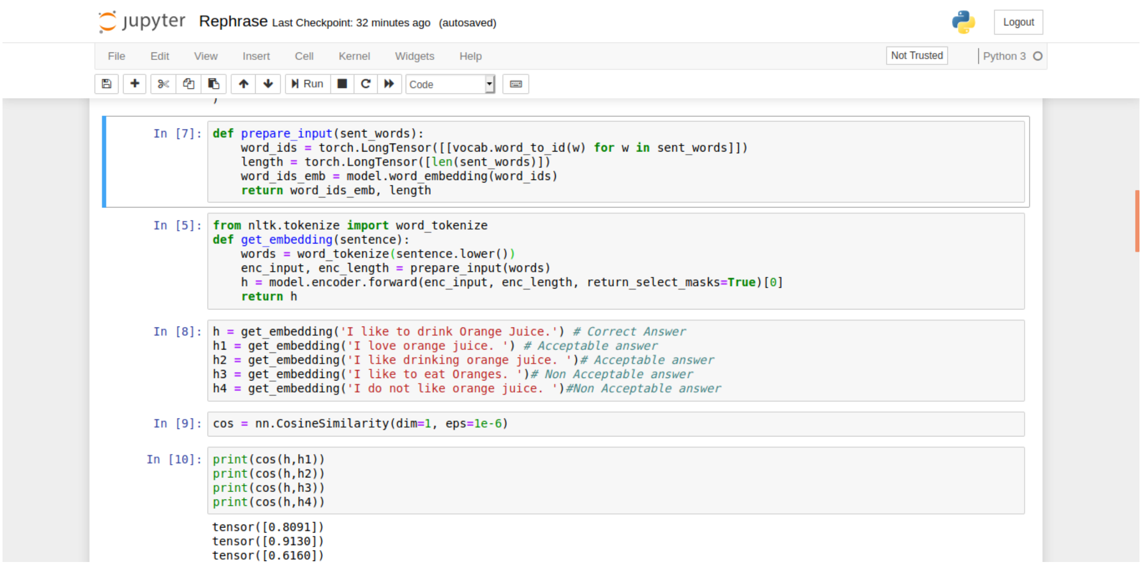
```
atul@atul-sahay:~/Disk1/work/Upwork/Semantic-Code-Matcher$ pip3 install -r Requi
rements.txt
Collecting absl-py==0.9.0 (from -r Requirements.txt (line 1))
^CERROR: Operation cancelled by user
WARNING: You are using pip version 19.2.2, however version 20.0.2 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
atul@atul-sahay:~/Disk1/work/Upwork/Semantic-Code-Matcher$
```

Figure 1: Installation of the required modules

3 Running the Pretrained Model

For the sake of running the pretrained model, all you need to is to open the jupyter notebook **Rephrase.ipynb**. See Fig : 3

user: Explainable-latent-structures$: **jupyter notebook Rephrase.ipynb**



```
In [7]: def prepare_input(sent_words):
            word_ids = torch.LongTensor([[vocab.word_to_id(w) for w in sent_words]])
            length = torch.LongTensor([len(sent_words)])
            word_ids_emb = model.word_embedding(word_ids)
            return word_ids_emb, length

In [5]: from nltk.tokenize import word_tokenize
        def get_embedding(sentence):
            words = word_tokenize(sentence.lower())
            enc_input, enc_length = prepare_input(words)
            h = model.encoder.forward(enc_input, enc_length, return_select_masks=True)[0]
            return h

In [8]: h = get_embedding('I like to drink Orange Juice.') # Correct Answer
        h1 = get_embedding('I love orange juice. ') # Acceptable answer
        h2 = get_embedding('I like drinking orange juice. ')# Acceptable answer
        h3 = get_embedding('I like to eat Oranges. ')# Non Acceptable answer
        h4 = get_embedding('I do not like orange juice. ')#Non Acceptable answer

In [9]: cos = nn.CosineSimilarity(dim=1, eps=1e-6)

In [10]: print(cos(h,h1))
         print(cos(h,h2))
         print(cos(h,h3))
         print(cos(h,h4))

        tensor([0.8091])
        tensor([0.9130])
        tensor([0.6160])
```

Figure 2: Jupyter Notebook

4 Data Load and Preprocessing

All these operations will be done in **Extraction-latent-structures-by-Using-Attention/snli** directory. change your directory as given in the Section 1[Project Directory]

4.1 Data Download

For the data loading and data preprocessing, you need to follow the given mentioned steps:

1. You need to download the SNLI data and Glove embeddings and store in the repective **snli** directory.
   - https://nlp.stanford.edu/projects/snli/
   - http://nlp.stanford.edu/data/glove.840B.300d.zip
2. Now just extract them.

4.2 Building Vocabulary

For building the vocabulary just run the below command while be in the **Extraction-latent-structures-by-Using-Attention/snli** directory. Change your directory as given in the Section 1. You are going to use build_vocab.py file for this operation.

$-->$ **$ python3 build_vocab.py –data-paths file1;file2;file3 –out Vocab/vocab.pkl**

Following are the arguments:

1. –data-paths file1, file 2 and file3 are separated by ; basically indicates the files of which you want to build vocab for the model. For say if I want to build vocab for train.jsonl and test.jsonl I will pass it as train.jsonl;test.jsonl
2. –out Vocab/vocab.pkl to store the vocab file.

4.3 Making a Dataset File

Change the directory to project parent Directory as **Extraction-latent-structures-by-Using-Attention**. You are going to use the dump_dataset.py for this operation.

Please run the following command each time for files train.jsonl, valid.jsonl, test.jsonl [You can access all of these files from the snli dataset you have downloaded in section 4.1]

$-->$ **$ python3 dump_dataset.py –data file1 –vocab snli/Vocab/vocab.pkl –vocab-size 100000 –max-length 200 –out Data/train.pkl**

Following are the arguments:

1. –data file1 is the file of you want the dataset object for say you want to create a dataset object for the test.jsonl file then pass the file location as the file1 [Note : you are in project home directory so pass the location with reference to it.]
2. –out where you want to store the dataset class [Do store all such files in the Data Folder see Section 1 for the project directory]

3. vocab-size Maximum limit of the vocab you want to use. [Note : Ignore this]
4. max-length Maximum length of the sentence you want to use. [Note : Ignore this]

5 Training The Model

Change the directory to project parent Directory as **Extraction-latent-structures-by-Using-Attention**. You are going to use the train.py for this operation.

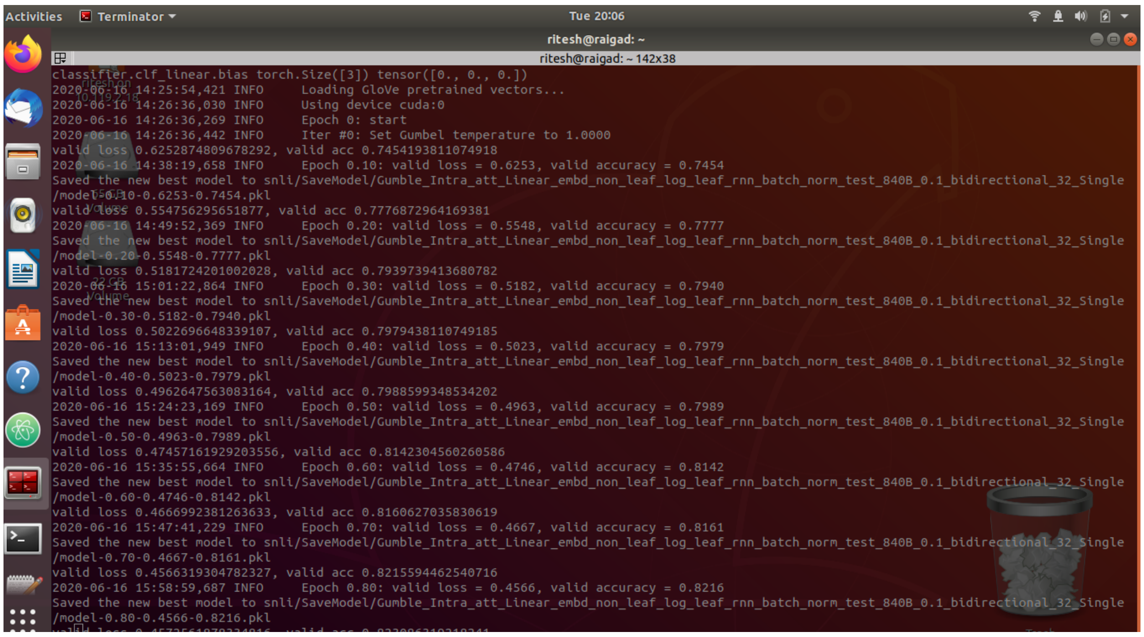
5.1 Model Characteristics

- The whole model is trained on the GPU specification:
  - Name: Geforce RTX 2080 ti
  - Memory: 11 GB
  - Cuda version 10.1
- The whole process takes around 2-3 days for completion

Run the below command for training the model.

$--\ >$ **$ CUDA_LAUNCH_BLOCKING=1 python3 train.py –train-data snli/Data/train.pkl –valid-data snli/Data/val.pkl –glove snli/glove.840B.300d.txt –save-dir snli/SaveModel/ –word-dim 300 –hidden-dim 300 –clf-hidden-dim 1024 –clf-num-layers 1 –batch-size 64 –max-epoch 20 –device cuda –intra-attention –leaf-rnn –dropout 0.1**

Once the training is completed you will have a number of model files which you can load in Rephrase.ipynb to get the embedding and use it in cosine similarity.



```
classifier.clf_linear.bias torch.Size([3]) tensor([0., 0., 0.])
2020-06-16 14:25:54,421 INFO     Loading GloVe pretrained vectors...
2020-06-16 14:26:36,030 INFO     Using device cuda:0
2020-06-16 14:26:36,269 INFO     Epoch 0: start
2020-06-16 14:26:36,442 INFO     Iter #0: Set Gumbel temperature to 1.0000
valid loss 0.6252874809678292, valid acc 0.7454193811074918
2020-06-16 14:38:19,658 INFO     Epoch 0.10: valid loss = 0.6253, valid accuracy = 0.7454
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.10-0.6253-0.7454.pkl
valid loss 0.554756295651877, valid acc 0.7776872964169381
2020-06-16 14:49:52,369 INFO     Epoch 0.20: valid loss = 0.5548, valid accuracy = 0.7777
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.20-0.5548-0.7777.pkl
valid loss 0.5181724201002028, valid acc 0.7939739413680782
2020-06-16 15:01:22,864 INFO     Epoch 0.30: valid loss = 0.5182, valid accuracy = 0.7940
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.30-0.5182-0.7940.pkl
valid loss 0.5022696648339107, valid acc 0.7979438110749185
2020-06-16 15:13:01,949 INFO     Epoch 0.40: valid loss = 0.5023, valid accuracy = 0.7979
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.40-0.5023-0.7979.pkl
valid loss 0.4962647563083164, valid acc 0.7988599348534202
2020-06-16 15:24:23,169 INFO     Epoch 0.50: valid loss = 0.4963, valid accuracy = 0.7989
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.50-0.4963-0.7989.pkl
valid loss 0.47457161929203556, valid acc 0.8142304560260586
2020-06-16 15:35:55,664 INFO     Epoch 0.60: valid loss = 0.4746, valid accuracy = 0.8142
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.60-0.4746-0.8142.pkl
valid loss 0.4666992381263633, valid acc 0.8160627035830619
2020-06-16 15:47:41,229 INFO     Epoch 0.70: valid loss = 0.4667, valid accuracy = 0.8161
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.70-0.4667-0.8161.pkl
valid loss 0.4566319304782327, valid acc 0.8215594462540716
2020-06-16 15:58:59,687 INFO     Epoch 0.80: valid loss = 0.4566, valid accuracy = 0.8216
Saved the new best model to snli/SaveModel/Gumble_Intra_att_Linear_embd_non_leaf_log_leaf_rnn_batch_norm_test_840B_0.1_bidirectional_32_Single
/model-0.80-0.4566-0.8216.pkl
```

Figure 3: Training Operation taking place

6 Architectural Details

6.1 Tree Lstm

Tree-structured long short-term memory network (Tree-LSTM) is an elegant variant of (Recursice RNN)RvNN, where it controls information flow from children to parent using similar mechanism to long short-term memory (LSTM) (Hochreiterand Schmidhuber 1997). Tree-LSTM introduces cell state in computing parent representation, which assists each cell to capture distant vertical dependencies.

$$\begin{bmatrix} i \\ f_l \\ f_r \\ o \\ g \end{bmatrix} = \begin{bmatrix} \sigma \\ \sigma \\ \sigma \\ \sigma \\ tanh \end{bmatrix} \left(W_{comp} \begin{bmatrix} h_l \\ h_r \end{bmatrix} + b_{comp} \right) \tag{1}$$

$$c_p = f_l \odot c_l + f_r \odot c_r + i \odot g \tag{2}$$

$$h_p = o \odot tanh(c_p) \tag{3}$$

where $W_{comp} \epsilon R^{5D_h \times 2D_h}, b_{comp} \epsilon R^{2D_h}$ and $\odot$ is the element wise product .

6.2 Parent Selection

Since information about the tree structure of an input is not given to the model See Figure **??**, a special mechanism is needed for the model to learn to compose task-specific tree structures in an end-to-end manner. We now describe the mechanism for building up the tree structure from an unstructured sentence.

First, our model introduces the trainable composition query vector $q \epsilon R_h^D$. The composition query vector measures how valid a representation is. Specifically, the validity score of a representation $r = [h; c]$ is defined by $q\dot{h}$.

At layer t, the model computes candidates for the parent representations using Eqs. 1–3. Then, it calculates the validity score of each candidate and normalize it so that $\sum_{i=1}^{M_{t+1}} v_i = 1$

$$v_i = \frac{\exp(q\dot{h}_i^{t+1})}{\sum_{j=1}^{M_{t+1}} \exp(q\dot{h}_j^{t+1})} \tag{4}$$

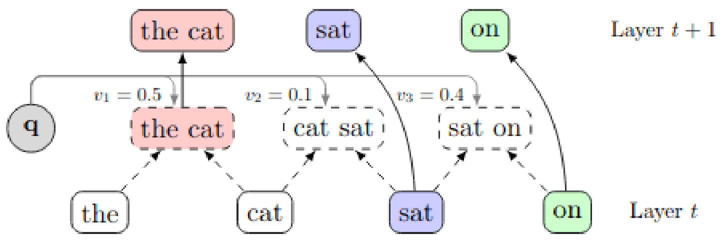


Figure 4: An example of the parent selection. At layer t(the bottom layer), the model computes parent candidates (the middle layer). Then the validity score of each candidate is computed using the query vector q(denoted as v_1, v_2, v_3). In the training time, the model samples a parent node among candidates weighted on v_1, v_2, v_3, using Softmax estimator, and in the testing time the model selects the candidate with the highest validity. At layer t+ 1(the top layer), the representation of the selected candidate ('the cat') is used as a parent, and the rest are copied from those of layer t('sat', 'on'). Best viewed in color.

6.3 Attention

.

We focused on type specific attention mechanism that aims to encourage the model to focus on salient latent information of the composition or constituency tree that is relevant for the classification decision We denoted the output of the intermediate nodes and leaf nodes in the layers of the Tree LSTM as $(\overrightarrow{h_1}, \overrightarrow{h_2}, \cdots, \overrightarrow{h_{2n-1}})$ where 'n' is the sentence length.

Now, the formulation of the architecture as follows :

$$\tilde{a_i} = \exp(W_a \times h_i) \tag{5}$$

$$a_i = \frac{\tilde{a_i}}{\sum_{i=1}^{2n-1}(\tilde{a_i})} \tag{6}$$

$$\vec{H_c} = \sum_{i=1}^{2n-1}(a_i \vec{h_i}) \tag{7}$$

In our architecture, the context vector H_c is learnt using attention weighted contextual representation by passing Node vectors to the fully connected layer with D_a hidden units in a hidden layer,,. For each such hidden state vector e_i, a scalar unit a_i is learnt with the weight matrix $W_a \epsilon R^{1 \times D_a}$. These scalar unit are then normalized to 1 and the whole formulation of context word vector is defined in Eq. 7.

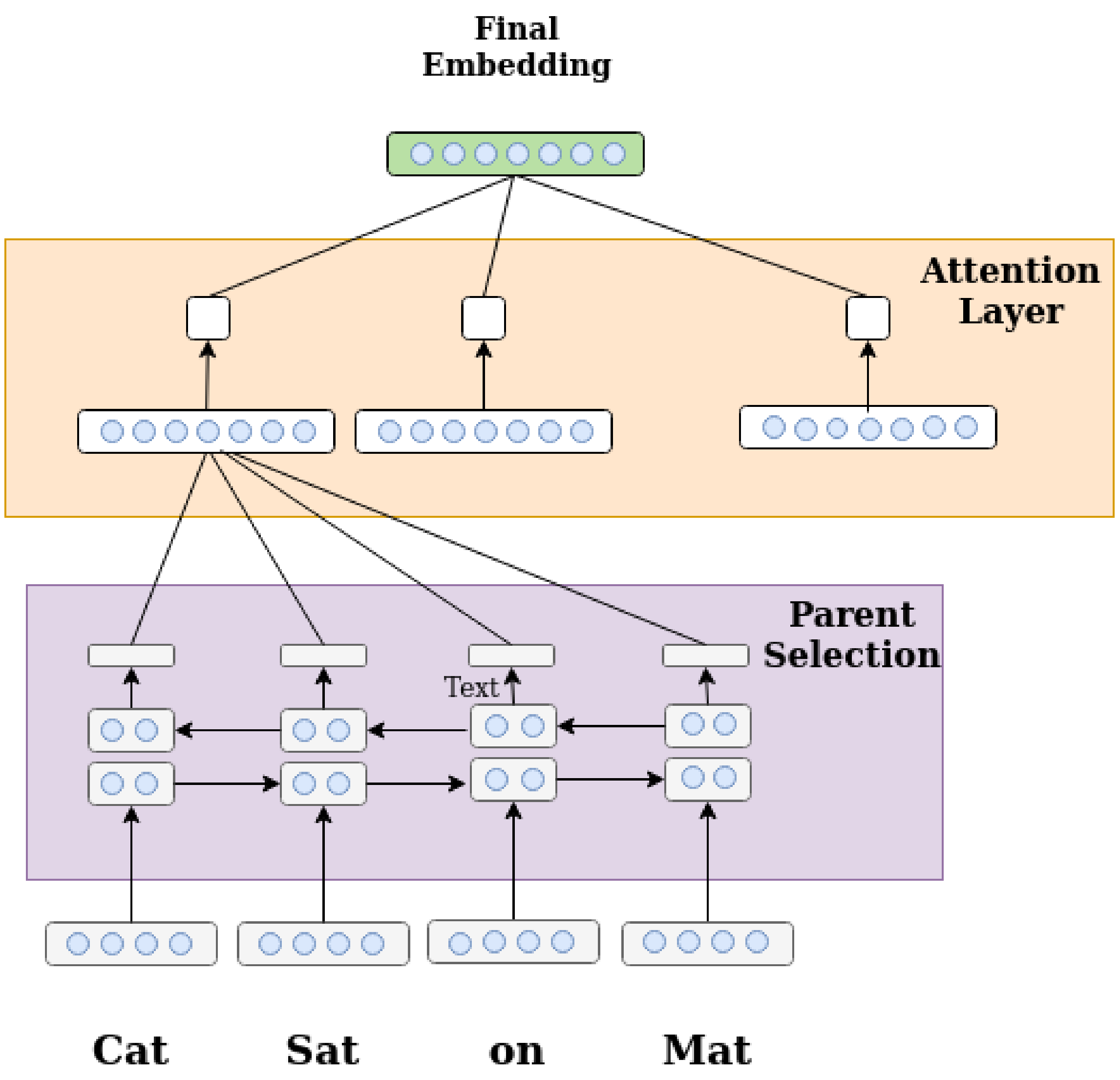


Figure 5: Model's Architecture