# Techniques and resources used

After prepossessing, the next step is to build machine learning models. We have established three models, which use three different deep learning networks. The three types of neural networks are Word2vec[1], Convolutional Neural Networks(CNN) and Recurrent Neural Networks(RNN)[3]. In this section, we first briefly introduce how these networks work and then demonstrate how they make up our models.

## Word2vec

### Embedding Layer

The basic idea of our model is applying Word2vec method to transform the comment and code to vectors which machine learning models can understand. Therefore, the first layer of the three models must be Embedding Layer. As the first layer of the model, the embedding layer will receive the input data and map them to vectors. An embedding is a mapping of a discrete, categorical variable to a vector of continuous numbers. The purpose of Embedding Layer ist embeds high-dimensional word vectors into a low-dimensional space. The vector space representation of words has two properties: (1) semantically similar words are very close in the resulting vector space, and (2) the direction of both grammatical, as well as semantic relations between words, remains stable for different pairs of words.

***The example can be dropped from the report, but I will use it for presentation.***

(For example, if our data contains 3 vocabularies, the matrix will be the following. It is a 3\*3 matrix, the words have no relation with each other.

books = ["War and Peace", "Anna Karenina",

"The Hitchhiker's Guide to the Galaxy"]

books\_encoded = [[1, 0, 0],

[0, 1, 0],

[0, 0, 1]]

The embedding layer will map the matrix to lower dimension matrix:

books = ["War and Peace",

"Anna Karenina",

"The Hitchhiker's Guide to the Galaxy"]

books\_encoded\_ideal = [[0.53, 0.85],

[0.60, 0.80],

[-0.78, -0.62]

Similarity (dot product) between First and Second = 0.99

Similarity (dot product) between Second and Third = -0.94

Similarity (dot product) between First and Third = -0.97

The new Matrix is 3\*2 that means for each vocabulary it has a coordinate. Then we can measure the distance of each vocabulary. From the distance of each two vocabulary, we can estimate the similarity of them. The higher the value, the more similar they are.)

## Convolutional Neural Networks

We pick up the idea from Text-CNN[2] models for the project. There are two special layers in Text-CNN models, they are convolutional layer and the pooling layer. The core function of them in the text classification model is the feature extraction. From the input fixed-length text sequence, the local word is used to extract the primary features, and the primary features are combined into advanced features. With convolution and pooling operations, the steps of feature engineering in traditional machine learning can be eliminated.

But an obvious shortcoming of Text-CNN is that the convolution and pooling operations lose the order and position information of the words in the text sequence, and it is more difficult to capture the semantic information such as negation and antisense in the text sequence.

## Recurrent Neural Networks

As we discussed above, Test-CNN can only take and process each input individually. The previous input and the next input are completely unrelated. However, some tasks need to be able to better process the sequence information, that is, the previous input and the subsequent input are related. RNN can realize it. RNN is actually a neural network, but there is only one more hidden state to save historical information. In our project, we utilize the Gated Recurrent Units(GRU) layer and Attention mechanism.

### Gated Recurrent Units layer

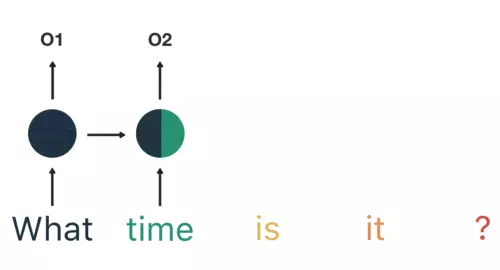
GRU is a variation on the RNN layer. The input of it is not the whole sentence anymore, but a single word. The GRU will remember it's important information and forget the useless information on the hidden state. That information will merge with the information of the next word and do remembering and forgetting again. Remember the old data which can influence the prediction result. This is the most important mechanism for manipulating streaming data (such as text or audio)

***The example can be dropped from the report, but I will use it for presentation.***

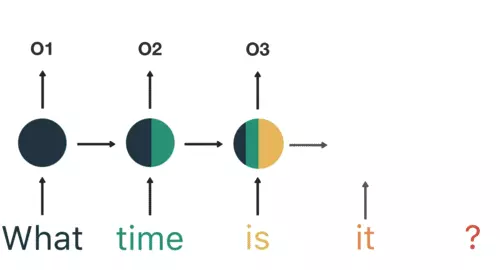
For example, we want to work with the following sentence, the sentence should be separated into single words. RNNs can only enter one word at a time.



The first step is to input "What" into the RNN, the RNN encodes "what" and produces the output. And then we provide the word "time" and the hidden state in the previous step. RNN now has information about the two words "what" and "time".



We repeat this process until the last step. In the last step that RNN encodes the information of all the words in the previous step.



### Attention layer

When we think about the English word “Attention”, we know that it means directing focus at something and taking greater notice. The Attention mechanism in Deep Learning is based on this concept of directing focus, and it pays greater attention to certain factors when processing the data. What the Attention component of the network will do for each word in the output is map the important and relevant words from the input sentence and assign higher weights to these words, enhancing the accuracy of the output prediction.

For example, if we want to predict whether the sentence “What time is it?” is an Interrogative sentence, the word “What” is more important than the other 3 words. The attention layer will assign higher weights to word “What” and assign lower weights to others.

## Model building and Experiments

Accoding to previous discussions, we built 3 models applying different layers. For all models, the first layer should be embedding layer. The comments and corresponding code are merged together and convert to vectors by one-hot encoding. The length of All training sentence are padded to 50.

### Embedding model

The first model is a single embedding layer with an output layer. The model structure the following.

Layer (type) Output Shape Param #

=================================================================

embedding (Embedding) (None, 50, 32) 64000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 1600) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 1) 1601

=================================================================

Total params: 65,601

Trainable params: 65,601

Non-trainable params: 0

Actually, after embedding layer we apply Logistic Regression to predict the result.

### CNN model

Layer (type) Output Shape Param #

=========================================================================

x\_seq (InputLayer) (None, 50) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

embedding (Embedding) (None, 50, 32) 64000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv1d (Conv1D) (None, 49, 10) 650

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

conv1d (Conv1D) (None, 48, 10) 970

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling1d (MaxPooling1D) (None, 1, 10) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

max\_pooling1d (MaxPooling1D) (None, 1, 10) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 10) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 10) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

concatenate (Concatenate) (None, 20) 0 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dropout (Dropout) (None, 20) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 16) 336

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 1) 17

=========================================================================

Total params: 65,973

Trainable params: 65,973

Non-trainable params: 0

For Cnn model, we set two convolutional-pooling layers and concatenate their result before the output layer.

### RNN model

Layer (type) Output Shape Param #

=================================================================

embedding (Embedding) (None, 50, 32) 64000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

gru (GRU) (None, 50, 16) 2352

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

seq\_self\_attention (None, 50, 16) 1089

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

flatten (Flatten) (None, 800) 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

dense (Dense) (None, 1) 801

=================================================================

Total params: 68,242

Trainable params: 68,242

Non-trainable params: 0

The RNN model is similar to embedding model but between embedding and output layer we add GRU and attention layer.

The trainings were conducted on a computer (Intel i5, 8GB RAM) with 1 GPU( NVIDIA GeForce GTX 1660).

[1]Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations

in vector space. arXiv preprint arXiv:1301.3781 (2013)

[2]Yoon Kim. York University. Convolutional Neural Networks for Sentence ClassificationNew. arXiv:1408.5882v2 [cs.CL] 3 Sep 2014

[3]<https://karpathy.github.io/2015/05/21/rnn-effectiveness/>