Q1: What is EUREKA-MangoNMT?
A1: EUREKA-MangoNMT is a neural machine translation toolkit, which is written in C++ within CPU platform. It is extended from EUREKA which is written by Ashish Vaswani. EUREKA implements a basic encoder-decoder framework that uses a left-to-right LSTM to encode a sequence into a real-valued vector from which the decoder uses another left-to-right LSTM to generate the output sequence. NCE is used for network training. EUREKA-MangoNMT has several key new features: 1) it adds rich code comments for important classes and functions; 2) it implements the bidirectional LSTMs to encode the input sequence; 3) it implements the global attention mechanism which attends different source contexts when generating outputs in different positions; 4) it implements the feed input mechanism which links the attention output to the next decoder LSTM node; 5) it supports training data shuffle in different training epoch; 6) it implements a novel NMT framework: one sentence one model for neural machine translation.

Q2: What about the translation performance?
A2: EUREKA-MangoNMT can obtain the similar translation performance to the Theano-based DL4MT.

Q3: What about the training time:
A3: It highly depends on the power of the CPU you use. In our experiments using 24 threads on Intel(R) Xeon(R) CPU E5-2690 @ 2.90GHz, it takes about one week to training the NMT model on about 700k LDC parallel sentence pairs. In this experiments, the input embedding and encoding hidden dimension are set 256, while the output embedding and decoding hidden dimension are set 512. It is slightly slower than Theano-based DL4MT using GPUs.

Q4: What is neural machine translation?
A5: Neural machine translation is a new paradigm for machine translation using deep neural networks. In contrast to statistical machine translation that employs a log-linear model $p(y|x) \propto \sum_i \lambda_i f_i(x,y)$ to integrate multiple sub-model features (e.g. translation model, language model and reordering model), neural machine translation employs deep learning and directly models the conditional probability $p(y|x)$ by using follow formulae:

$$p(y|x) = \prod_i p(y_i|y_{<i}, x) = \prod_i p(y_i|y_{<i}, C)$$

In which C denotes the source-side context representation. In this code implementation, C is obtained by the encoder through the bidirectional LSTMs. The left-to-right LSTM learns the forward context representation $C_f = (\vec{h}_1, \vec{h}_2, \cdots, \vec{h}_j, \cdots, \vec{h}_{T_x})$ as follows:

$$\vec{h}_j = LSTM(\vec{h}_{j-1}, x_j)$$

The backward context representation C_b can be learnt similarly. Then the

representation for each source-side word x_j is denoted by $h_j = [\overrightarrow{h_j}; \overleftarrow{h_j}]$.

The decoder calculates the conditional probability $p(y_i|y_{<i}, C) = p(y_i|y_{<i}, c_i) = g(y_i, \hat{z}_i, c_i)$ in which $\hat{z}_i$ is the attention output state at position i and is computed as follows:

$$\hat{z}_i = tahn(z_i, c_i)$$

In which z_i is the decoder hidden state at position I and is computed with the decoder LSTM node:

$$z_i = LSTM(z_{i-1}, y_{i-1}, \hat{z}_{i-1})$$

c_i is learnt by weighting all the source-side context vectors $C = (h_1, h_2, \cdots, h_j, \cdots, h_{T_x})$. $g(y_i, \hat{z}_i, c_i) = softmax(y_i \cdot \hat{z}_i)$.

The encoding bidirectional LSTM and decoding LSTM can be illustrated in the following figure.

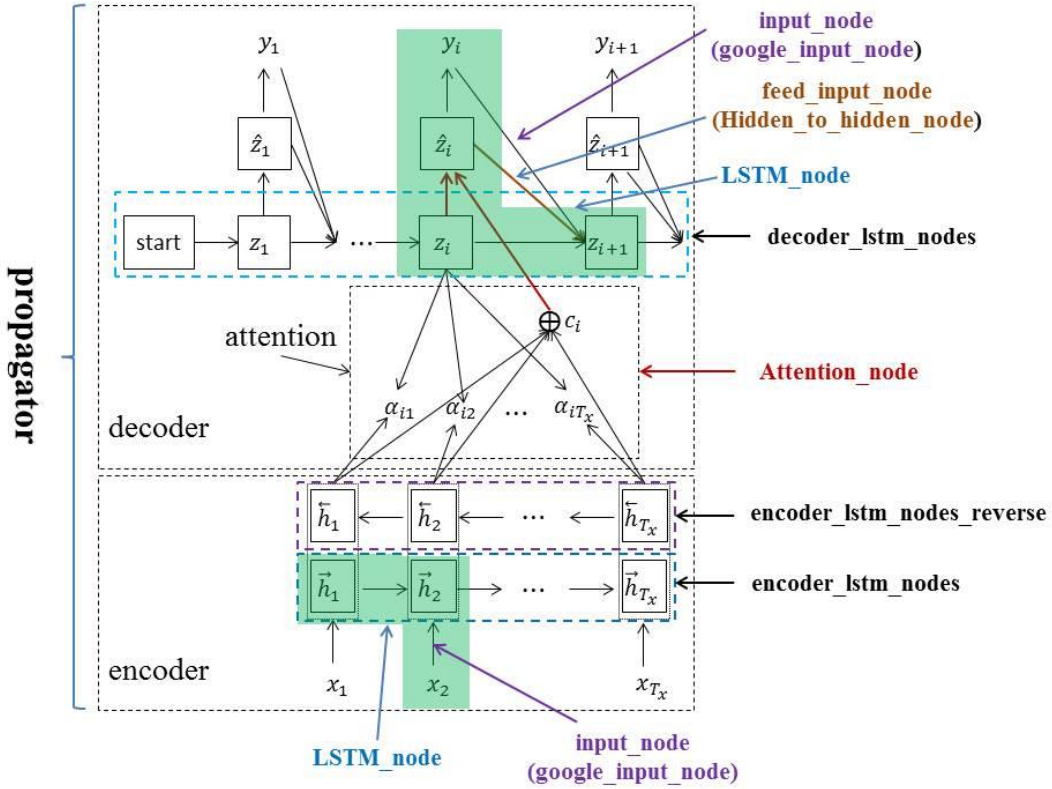

Fig. 1: Model structure of EUREKA-MangoNMT system.

The model structure and its relations with the C++ code organization are illustrated as follows:

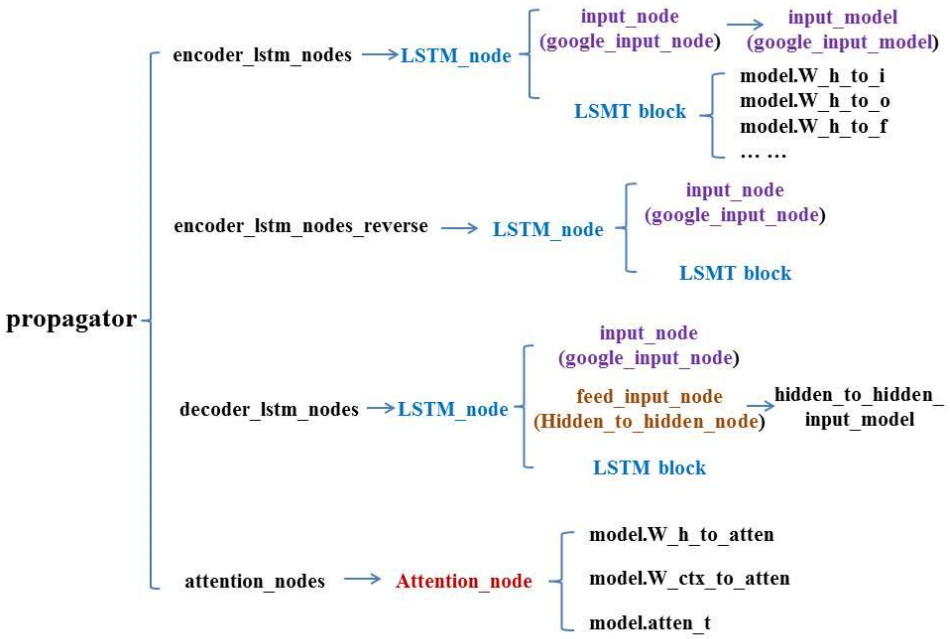

Fig 2. The relations between different objects and classes

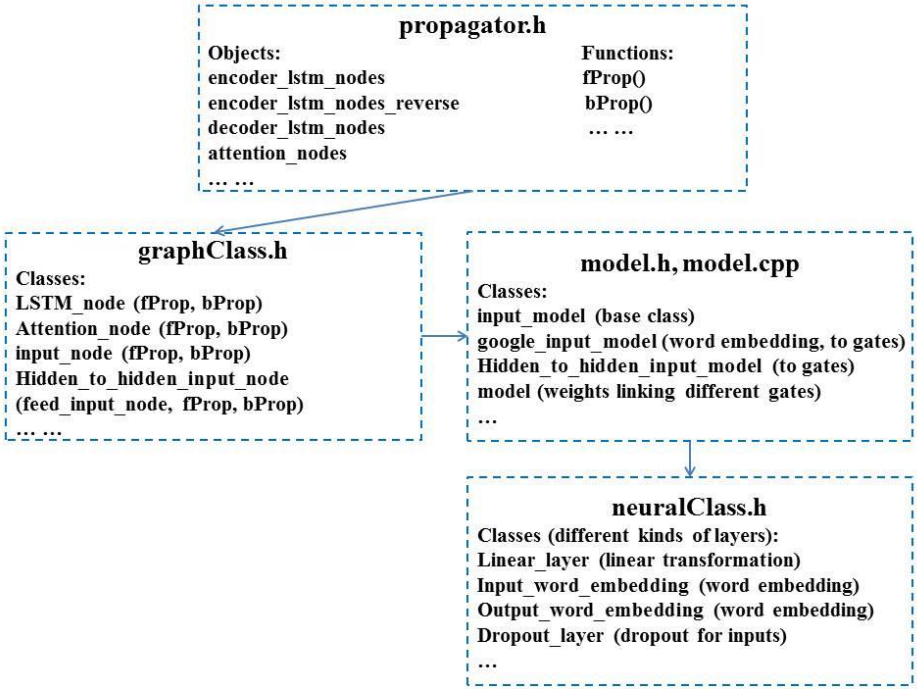

Fig. 3: Relations between different code files

The core functions are implemented in "propagator.h", in which left-to-right and right-to-left source-side context representations are learnt through encoder_lstm_nodes and encoder_lstm_nodes_reverse with LSTM_node sequence using forward propagation (fProp); decoding from the source-side context representations is implemented by decoder_lstm_nodes with LSTM node and attention_nodes with Attention_node (fProp). Backpropagation functions are

also defined in this file.

Different nodes including LSTM_node, input_node and Attention_node are implemented in “graphClass.h”. All the nodes used in encoder_lstm_nodes (encoder_lstm_nodes_reverse, decoder_lstm_nodes, or attention_nodes) apply the same model parameters.

Different models are implemented in “model.h” and “model.cpp”. All the models define network parameters which are linear transformation and non-linear transformation (defined using layers).

Different kinds of layers are defined in “neuralClass.h”, in which linear transformation, non-linear transformation, word embedding handling and dropout are defined.

TODO:
1, Implementing BlackOut algorithm to speed up the training procedure.
2, ensemble decoding