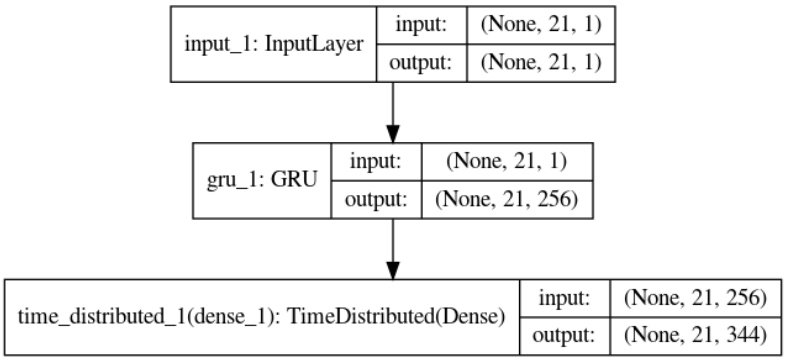
Assignment#2: Machine Translation

Model Architecture Tested and Used

Keng Hin Cheong

Model 1-Simple RNN

The graph of the architecture:



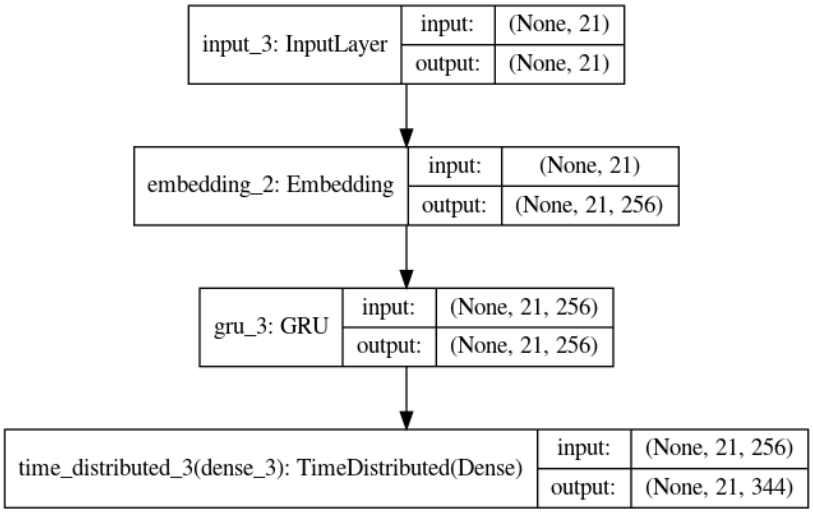
The code:

```
# build and train a basic RNN on x and y
def simple_model(input_shape, output_sequence_length, english_vocab_size, french_vocab_size):

    # build the layers and set up parameters to prepare the model to be trained
    learning_rate = 1e-3

    input_seq = Input(input_shape[1:])
    rnn = GRU(256, return_sequences=True)(input_seq)
    logits = TimeDistributed(Dense(french_vocab_size, activation='softmax'))(rnn)

    model = Model(inputs=input_seq, outputs=logits)
    model.compile(loss=sparse_categorical_crossentropy,
                  optimizer=Adam(learning_rate),
                  metrics=['accuracy'])
    return model
```

Explanation:

The layer 1 of the model uses an GRU module with english_vocab_size hidden units. With setting return_sequences to True, return not only the last output, but also all the outputs so far in the form of (num_samples, timesteps, output_dim). This is needed, because TimeDistributed in the below expects the first dimension to be the timesteps. Afterwards, apply a dense layer to the every temporal slice of an input.

Model 2-RNN with Embedding
The graph of the architecture:

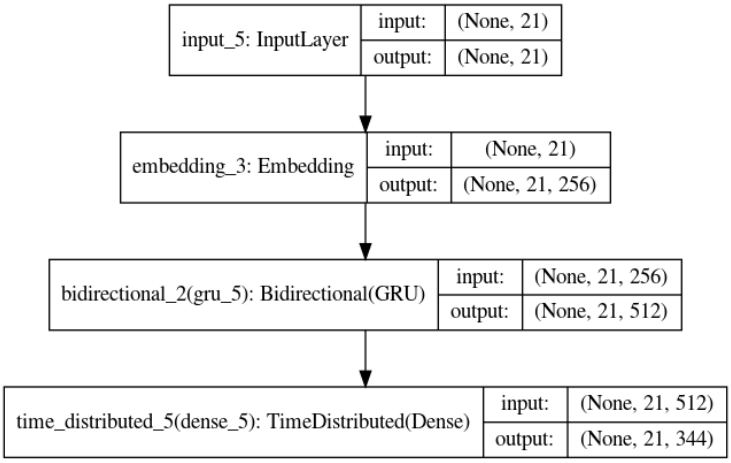


The code:

```
# build and train a RNN model using word embedding on x and y
def embed_model(input_shape, output_sequence_length, english_vocab_size, french_vocab_size):
    # Build the layers and set up parameters to prepare the model to be trained
    learning_rate = 1e-3

    input_seq = Input(input_shape[1:])
    embed_layer = Embedding(english_vocab_size, 256, input_length=output_sequence_length)(input_seq)
    rnn = GRU(256, return_sequences=True)(embed_layer)

    logits = TimeDistributed(Dense(french_vocab_size, activation='softmax'))(rnn)

    model = Model(inputs=input_seq, outputs=logits)
    model.compile(loss=sparse_categorical_crossentropy,
                  optimizer=Adam(learning_rate), metrics=['accuracy'])
    return model
```

Explanation:
The layer 1 of the model uses embedding layer to help enhance the representation of the word. The layer 2 uses an GRU module with english_vocab_size hidden units. Afterwards, apply a dense layer to the every temporal slice of an input.

Model 3-Bidirectional RNN with Embedding
The graph of the architecture:



The code:

```
# build a model that incorporates embedding, and bidirectional RNN on x and y
def Bd_Emb_model(input_shape, output_sequence_length, english_vocab_size, french_vocab_size):

    # build the layers and set up parameters to prepare the model to be trained
    learning_rate=1e-3

    input_seq = Input(input_shape[1:])
    emb = Embedding(english_vocab_size, 256, input_length=output_sequence_length)(input_seq)
    bdrnn = Bidirectional(GRU(256, return_sequences=True))(emb)
    logits = TimeDistributed(Dense(french_vocab_size, activation='softmax'))(bdrnn)

    model = Model(inputs=input_seq, outputs=logits)
    model.compile(loss=sparse_categorical_crossentropy,
                  optimizer=Adam(learning_rate),
                  metrics=['accuracy'])

    return model
```

Explanation:
The layer 1 of the model uses embedding layer to help enhance the representation of the word. The layer 2 uses a Bidirectional wrapper and a GRU layer, and the layer 3 uses Bidirectional wrapper. At last, layer 4 is a dense layer that is applied to every temporal slice of an input.