1 Questions :

1.1 Question 1 :

Attention of y over x :

- Q : y
- K : x
- V : x

Attention of x over x :

- Q : x
- K : x
- V : x

1.2 Question 2 :

When computing attention, we perform the product QK^T with a matrix V (we ommit the softmax operation here). The cost of multiplying a (nxd) matrix (Q) with a (dxn) matrix (K transposed) is n^2d. Then the cost of multiplying the (nxn) matrix we got with the (nxd) matrix (V) is again n^2d. Thus the complexity per attention layer is n^2d.

In recurrent models, the hidden state matrix is of size d^2. Thus, the cost of computing an input sentence of size n is hence nd^2.

The layers of convolutional models are the same as the recurrent models except that we look only at k elements of the input sentence, and then we we translate the kernel by one word and look at k elements again, and this is done n times until we've went through all the words of the sentence. Thus the complexity here is knd^2.

1.3 Question 3 :

In addition to the reason of being computationally effective, having multiple heads of attention allows the decoder to focus on multiple parts and aspects of the input sequence. (For instance some heads focus on the key words of the sequence whereas other heads focus on the grammatical and/or semantic relations between them). Plus using this technique we obtain multiple sets of Query/Key/Value weight matrices which each one of these matrices is randomly initialized. Thus it gives the attention layer multiple representation sub-spaces of the input embedding.

1.4 Question 4 :

The sin and cos representation is nice since nearby positions have high similarity in their positional encodings, which may make it easier to learn transformations that "preserve" this desired closeness. Plus it can generalize to large sequences, even larger than those seen in the training data.

1.5 Question 5 :

This is a trick used to speed up training. Masking future words target and not using them to translate the current words by setting -∞ to the corresponding coefficients in QK^T makes these coefficients equal to 0 after going through the softmax function. Thus their "probability scores" that correspond to how much attention should be given to these words during the translation are null and are therefore not used by the decoder.

(Below -Figure 1- the attention mask shows the position each tgt word (row) is allowed to look at (column). Words are blocked for attending to future words during training.)

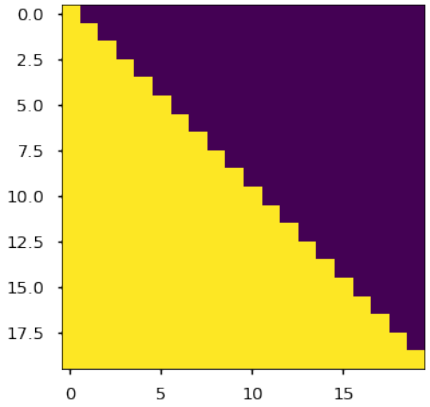

Figure 1: The mask used (the dark area corresponds to the coefficients set to 0 in $softmax(QK^T)$)

1.6 Question 6 :

There are 65M parameters in the base setting of this model and it has been trained on 8 GPUs for 12 hours. The answer is obviously no using my modest NVIDIA 1050Ti GPU.

1.7 Question 7 :

I did not manage to train completely the model (10 hours on my laptop / a lot less on colab but due to some internet issues it crashes midway). But I see that the model learns (the loss decreases). Plus I did not get to coding the part of the sequential prediction phase as I do not have a .pth up and ready. But you can see my code submitted with this report.

1.8 Question 8 :

BERT uses the encoder part of the Transformer to encode the sentence given as an input. The training of this encoding is performed on two different tasks (minimizing the sum of two different losses each corresponding to a task) :

- Predicting a masked word from the input sentence : Analogous to any standard word embedding learning tasks, the goal is to find a representation of a word as a vector. The attention mechanism serves the goal of training the net in a Bidirectional way, but in reality the sentences in the input are being fed all in the same time, thus the real mechanism is ”un-directional / non-directional” and the word being encoded pays attention to all the other words in the sequence.
- Prediction if sentence A and B are two successive sentences : The goal of this is empower its ability to generate coherent sentences and learn to catch ”far” dependencies in the sentences.

In the end of this training, we obtain word embeddings that are context dependent (But not only that since we can use other outputs of the model -such as the CLS token embedding- to perform another downstream task).

I would argue that this way of encoding (and thus the resulting representation) is efficient since we get word vectors that are context dependant, work for very long sentences and catch more types of dependencies between words in the sentence.

References

http://jalammar.github.io/illustrated-bert/
http://jalammar.github.io/illustrated-transformer/
https://colah.github.io/posts/2015-08-Understanding-LSTMs/
http://karpathy.github.io/2015/05/21/rnn-effectiveness/