MLP based Encoder:

For the MLP based encoder, a neural network with 3 hidden layers used. Instead of using ReLu, I have used Leaky ReLU. The only difference between ReLU and Leaky ReLU is that Leaky ReLU has a slope in the negative x axis, the slope is 0.01 by default but we can change. I have used the default 0.01 for my model. Leaky ReLU also brings little bit more non-linearity to the model. Hidden layer 2 and 3 are also drop out enabled with a drop out factor of 0.3. A ideal value for non-input layers is between 0.2 and 0.5. We can also use dropout for the input layer with a small drop out factor of about 0.2, but during my experimentation I did not see any significant increase in accuracy.

For the word embedding we used the standard glove.6B.300d word embedding. It contains 300 features for 6 billion words making it very robust in terms of representing a word. This embedding has higher accuracy while comparing it with glove.6B.50d or glove.6B.100d as they have lesser features.

In my model I have used a learning rate of 0.01, we have to tune this hyper parameter by reducing in steps of 10^-1. We don't want out learning rate to be too small or else it will take a long to converge on the flip side we don't want it to be too large or else it will overshoot and may not reach the minima. In terms of optimizers, I have used Adam optimizer because it gives the best features of both momentum and RMSProp.

CNN based Encoder:

The idea of using CNN for NLP tasks is relatively new as CNN was mainly used in computer vision tasks. But we can apply it to text inputs by using a 1D filter. In case also we make use of the glove.6B.300d embedding due to the advantages mentioned above. I have used 100 filters on size 1 along max pooling. In the case of activation function, I have used Leaky ReLU again. Post that the outputs are fed to a max pooling layer. At the end we will have a column vector with 100 elements. This is subjected to a fully connected layer.

This above step is done to both sentence A and sentence B of the input and post that we apply the cosine similarity function to it. The higher the number we get the more similar the two sentences are.

In terms of learning rate and choice of optimizers it the same as for the MLP encoder. A learning rate of 0.01 is ideal in most cases and it not too small neither it is too large. And the Adam optimizer again under most circumstances produces the best result. So that is the default choice going in.