BERT as Conditional Genrative model

Task:

To design Bert as conditional generative model, on giving condition it generate sentences according to that.

My approach to reach task:

- Took input from user.
- I found synonyms for given input words using NLTK library and ‘wordnet’ corpus.
- For each of this synonyms I tried to generate word.
- To use given word I put that word at any randomly masked position (forcefully by genrating random number).
- So when Bert try to generate sentence and try to fill the mask position, I bypassed the process when it comes to filling our position of conditional word.
- For this I have used code given in paper and modified it.
- I have modified mainly 2 function get_init_text & parallel_sequential_generation.
- And added one more function generate_statement to easily understand code and give input and see output.

Some Code Snippet:

1. putting condtional word at random position while intializing sentence.

```
def get_init_text(seed_text,rand_kk, max_len,conditional_word, batch_size = 1, rand_init=False):
    """ Get initial sentence by padding seed_text with either masks or random words to max_len """
    batch = [seed_text + [MASK] * (max_len) + [SEP] for _ in range(batch_size)]
    seed_len = len(seed_text)
    for jj in range(batch_size):
      batch[jj][seed_len + rand_kk] = conditional_word
    #if rand_init:
    #    for ii in range(max_len):
    #        init_idx[seed_len+ii] = np.random.randint(0, len(tokenizer.vocab))
    #print(batch)
    return tokenize_batch(batch)
```

2. checking and filling every masked words. (parallel_sequencial_generation)

```
def parallel_sequential_generation(seed_text,conditional_word, batch_size=10, max_len=15, top_k=0, temperature=None, max_iter=300, burnin=200,
                                   cuda=False, print_every=10, verbose=True):
    """ Generate for one random position at a timestep

    args:
        - burnin: during burn-in period, sample from full distribution; afterwards take argmax
    """
    rand_kk = np.random.randint(0,max_len)
    seed_len = len(seed_text)
    batch = get_init_text(seed_text,rand_kk, max_len,conditional_word, batch_size)

    for ii in range(max_iter):
        kk = np.random.randint(0, max_len)
        if(kk != rand_kk):
          for jj in range(batch_size):
              batch[jj][seed_len+kk] = mask_id
          inp = torch.tensor(batch).cuda() if cuda else torch.tensor(batch)
          out = model(inp)
          topk = top_k if (ii >= burnin) else 0
          idxs = generate_step(out, gen_idx=seed_len+kk, top_k=topk, temperature=temperature, sample=(ii < burnin))
          for jj in range(batch_size):
              batch[jj][seed_len+kk] = idxs[jj]

          if verbose and np.mod(ii+1, print_every) == 0:
              for_print = tokenizer.convert_ids_to_tokens(batch[0])
              for_print = for_print[:seed_len+kk+1] + ['(*)'] + for_print[seed_len+kk+1:]
              print("iter", ii+1, " ".join(for_print))

    return untokenize_batch(batch)
```

3. generating synonyms.

```
input_word= input()
synonyms = []

for syn in wordnet.synsets(input_word):
    for l in syn.lemmas():
        synonyms.append(l.name())

synonyms = list(set(synonyms))
print(set(synonyms))

sents = generate_statements(synonyms)
```