```
import argparse
import json
import os
# data: q, cq, (dq), (pq), y, *x, *cx
# shared: x, cx, (dx), (px), word_counter, char_counter, word2vec
# no metadata
from collections import Counter

from tqdm import tqdm

from squad.utils import get_word_span, get_word_idx, process_tokens


def main():
    args = get_args()
    prepro(args)


def get_args():
    parser = argparse.ArgumentParser()
    target_dir = "/Users/xavier.qiu/Documents/GitHub/bi-att-flow/data/squad"
    glove_dir = "/Users/xavier.qiu/Documents/GitHub/bi-att-flow/data/glove"  # os.p
    parser.add_argument('-s', "--source_dir", default=target_dir)
    parser.add_argument('-t', "--target_dir", default=target_dir)
    parser.add_argument('-d', "--debug", action='store_true')
    parser.add_argument("--train_ratio", default=0.9, type=int)
    parser.add_argument("--glove_corpus", default="6B")
    parser.add_argument("--glove_dir", default=glove_dir)
    parser.add_argument("--glove_vec_size", default=100, type=int)
    parser.add_argument("--mode", default="full", type=str)
    parser.add_argument("--single_path", default="", type=str)
    parser.add_argument("--tokenizer", default="PTB", type=str)
    parser.add_argument("--url", default="vision-server2.corp.ai2", type=str)
    parser.add_argument("--port", default=8000, type=int)
    parser.add_argument("--split", action='store_true')
    parser.add_argument("--version", default="1.1", action='store_true')

    # TODO : put more args here
    return parser.parse_args()


def create_all(args):
    out_path = os.path.join(args.source_dir, "all-v" + args.version + ".json")
    if os.path.exists(out_path):
        return
    train_path = os.path.join(args.source_dir, "train-v" + args.version + ".json")
    train_data = json.load(open(train_path, 'r'))
    dev_path = os.path.join(args.source_dir, "dev-v" + args.version + ".json")
    dev_data = json.load(open(dev_path, 'r'))
    train_data['data'].extend(dev_data['data'])
    print("dumping all data ...")
    json.dump(train_data, open(out_path, 'w'))
```

```python
def prepro(args):
    if not os.path.exists(args.target_dir):
        os.makedirs(args.target_dir)

    if args.mode == 'full':
        prepro_each(args, 'train', out_name='train')
        prepro_each(args, 'dev', out_name='dev')
        prepro_each(args, 'dev', out_name='test')
    elif args.mode == 'all':
        create_all(args)
        prepro_each(args, 'dev', 0.0, 0.0, out_name='dev')
        prepro_each(args, 'dev', 0.0, 0.0, out_name='test')
        prepro_each(args, 'all', out_name='train')
    elif args.mode == 'single':
        assert len(args.single_path) > 0
        prepro_each(args, "NULL", out_name="single", in_path=args.single_path)
    else:
        prepro_each(args, 'train', 0.0, args.train_ratio, out_name='train')
        prepro_each(args, 'train', args.train_ratio, 1.0, out_name='dev')
        prepro_each(args, 'dev', out_name='test')


def save(args, data, shared, data_type):
    """

    :param args:
    :param data:
    :param shared:
    :param data_type: train or dev
    :return:
    """
    data_path = os.path.join(args.target_dir, "data_{}.json".format(data_type))
    shared_path = os.path.join(args.target_dir, "shared_{}.json".format(data_type))
    json.dump(data, open(data_path, 'w'))
    json.dump(shared, open(shared_path, 'w'))


def get_word2vec(args, word_counter):
    glove_path = os.path.join(args.glove_dir, "glove.{}.{}d.txt".format(args.glove_
    sizes = {'6B': int(4e5), '42B': int(1.9e6), '840B': int(2.2e6), '2B': int(1.2e6
    total = sizes[args.glove_corpus]
    word2vec_dict = {}
    with open(glove_path, 'r', encoding='utf-8') as fh:
        for line in tqdm(fh, total=total):
            array = line.lstrip().rstrip().split(" ")
            word = array[0]
            vector = list(map(float, array[1:]))
            if word in word_counter:
                word2vec_dict[word] = vector
            elif word.capitalize() in word_counter:
                word2vec_dict[word.capitalize()] = vector
            elif word.lower() in word_counter:
                word2vec_dict[word.lower()] = vector
            elif word.upper() in word_counter:
                word2vec_dict[word.upper()] = vector
    print("{}/{} of word vocab have corresponding vectors in {}".format(len(word2ve
                                                                      glove_path)
    return word2vec_dict


def prepro_each(args, data_type, start_ratio=0.0, stop_ratio=1.0, out_name="default
    """

    :param args: configurations
    :param data_type: train or dev
    :param start_ratio:
    :param stop_ratio:
    :param out_name: train, dev, test
    :param in_path:
    :return:
    """
    if args.tokenizer == "PTB":
        import nltk
        sent_tokenize = nltk.sent_tokenize

        def word_tokenize(tokens):
            return [token.replace("''", '"').replace("``", '"') for token in nltk.w
    elif args.tokenizer == 'Stanford':
        from my.corenlp_interface import CoreNLPInterface
        interface = CoreNLPInterface(args.url, args.port)
        sent_tokenize = interface.split_doc
        word_tokenize = interface.split_sent
    else:
        raise Exception()

    if not args.split:
        sent_tokenize = lambda para: [para]

    # 1. load data
    source_path = in_path or os.path.join(args.source_dir, "{}-v{}.json".format(dat
    source_data = json.load(open(source_path, 'r'))
    # load the train  data or dev 1.1 dataset

    q, cq, y, rx, rcx, ids, idxs = [], [], [], [], [], [], []
    cy = []
    x, cx = [], []
    answerss = []
    p = []
    word_counter, char_counter, lower_word_counter = Counter(), Counter(), Counter(
    start_at_index = int(round(len(source_data['data']) * start_ratio))
    stop_at_index = int(round(len(source_data['data']) * stop_ratio))

    # for each article
    for article_index, article in enumerate(tqdm(source_data['data'][start_at_index
        xp, cxp = [], []
        pp = []
        x.append(xp)
        cx.append(cxp)
        p.append(pp)

        # for each paragrph of the article
        for paragraph_index, paragraph in enumerate(article['paragraphs']):
            # wordss
            context = paragraph['context']
            context = context.replace("''", '" ')
            context = context.replace("``", '" ')
            list_of_wordlist = list(map(word_tokenize, sent_tokenize(context)))
            list_of_wordlist = [process_tokens(tokens) for tokens in list_of_wordli
            # xi are words
            # given xi, add chars
            list_of_charlist = [[list(word) for word in word_list] for word_list in
            # cxi are characters for each words
            xp.append(list_of_wordlist)
            cxp.append(list_of_charlist)
            pp.append(context)

            # update the counter to plus the number of questions
            for wordlist in list_of_wordlist:
                for word in wordlist:
                    word_counter[word] += len(paragraph['qas'])
                    lower_word_counter[word.lower()] += len(paragraph['qas'])
                    for char in word:
                        char_counter[char] += len(paragraph['qas'])

            rxi = [article_index, paragraph_index]
            assert len(x) - 1 == article_index
            assert len(x[article_index]) - 1 == paragraph_index
            for question in paragraph['qas']:
                # get words
                question_wordslist = word_tokenize(question['question'])
                question_charslist = [list(qij) for qij in question_wordslist]
                yi = []
                cyi = []
                answers = []
                for answer in question['answers']:
                    answer_text = answer['text']
                    answers.append(answer_text)
                    answer_start = answer['answer_start']
                    answer_stop = answer_start + len(answer_text)
                    # TODO : put some function that gives word_start, word_stop her
                    yi0, yi1 = get_word_span(context, list_of_wordlist, answer_star
                    # yi0 = answer['answer_word_start'] or [0, 0]
                    # yi1 = answer['answer_word_stop'] or [0, 1]
                    assert len(list_of_wordlist[yi0[0]]) > yi0[1]
                    assert len(list_of_wordlist[yi1[0]]) >= yi1[1]
                    w0 = list_of_wordlist[yi0[0]][yi0[1]]
                    w1 = list_of_wordlist[yi1[0]][yi1[1] - 1]
                    i0 = get_word_idx(context, list_of_wordlist, yi0)
                    i1 = get_word_idx(context, list_of_wordlist, (yi1[0], yi1[1] -
                    cyi0 = answer_start - i0
                    cyi1 = answer_stop - i1 - 1
                    # print(answer_text, w0[cyi0:], w1[:cyi1+1])
                    assert answer_text[0] == w0[cyi0], (answer_text, w0, cyi0)
                    assert answer_text[-1] == w1[cyi1]
                    assert cyi0 < 32, (answer_text, w0)
                    assert cyi1 < 32, (answer_text, w1)
                    yi.append([yi0, yi1])
                    cyi.append([cyi0, cyi1])

                for qij in question_wordslist:
                    word_counter[qij] += 1
                    lower_word_counter[qij.lower()] += 1
                    for qijk in qij:
                        char_counter[qijk] += 1

                q.append(question_wordslist)
                cq.append(question_charslist)
                y.append(yi)
                cy.append(cyi)
                rx.append(rxi)
                rcx.append(rxi)
                ids.append(question['id'])
                idxs.append(len(idxs))
                answerss.append(answers)

            if args.debug:
                break

    word2vec_dict = get_word2vec(args, word_counter)
    lower_word2vec_dict = get_word2vec(args, lower_word_counter)

    # add context here
    data = {
        'q': q,  # list of word list of each questions, [['who','are', 'you'], ...
        'cq': cq,
        # [<class 'list'>: [['T', 'o'], ['w', 'h', 'o', 'm'], ['d', 'i', 'd'], ['t'
        'y': y,  # list of <class 'list'>: [[(0, 108), (0, 111)]]
        '*x': rx,  # list of <class 'list'>: [0, 21], 0 means the number of article
        '*cx': rcx,  # same with rx but for characters, i guess the values are same
        'cy': cy,  #
        'idxs': idxs,  # just those ids
        'ids': ids,  # the id of each question, sth like uuid
        'answerss': answerss,  # the content of the answer
        '*p': rx  #
    }
    shared = {
        'x': x,  # words of each paragraph
        'cx': cx,  # characters of each
        'p': p,  # the content of each paragraph
        'word_counter': word_counter,
        'char_counter': char_counter,
        'lower_word_counter': lower_word_counter,
        'word2vec': word2vec_dict,
        'lower_word2vec': lower_word2vec_dict
    }

    print("saving ...")
    save(args, data, shared, out_name)


if __name__ == "__main__":
    main()
```