dependency_parser.py

```
from __future__ import absolute_import
from __future__ import division
from __future__ import print_function

import gensim
import numpy as np
import tensorflow as tf
import argparse
import collections

parser = argparse.ArgumentParser()
parser.add_argument("--model", type=str,
                    default="dependency_parser_model", help="model
directory")
parser.add_argument("--batch", type=int, default=100, help="batch size")
parser.add_argument("--steps", type=int, default=10000, help="training
steps")
parser.add_argument('--train', action='store_true', help='with training')
parser.add_argument('--eval', action='store_true', help='with evaluation')
parser.add_argument('--predict', action='store_true', help='with
prediction')
args = parser.parse_args()

tf.logging.set_verbosity(tf.logging.INFO)


def dependency_parser_model_fn(features, labels, mode):
    """""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    dependency parser model
    implementation of "A Fast and Accurate Dependency Parser"
    (https://cs.stanford.edu/~danqi/papers/emnlp2014.pdf)

    features:   batch of features from input_fn
    labels:     batch of labels from input_fn
    mode:       enum { TRAIN, EVAL, PREDICT }
    """""""""""""""""""""""""""""""""""""""""""""""""""""""""""

    """""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    dense layer 1
    (-1, 8400) -> (-1, 1024)
    """""""""""""""""""""""""""""""""""""""""""""""""""""""""""

    dense1 = tf.layers.dense(
        inputs=features["x"],
        units=1024,
        activation=lambda x: tf.pow(x, tf.constant(3, dtype=tf.float64))
    )

    dense1 = tf.layers.batch_normalization(
        inputs=dense1,
        training=mode == tf.estimator.ModeKeys.TRAIN
    )

    """""""""""""""""""""""""""""""""""""""""""""""""""""""""""
    logits layer
    (-1, 1024) -> (-1, 10)
    """""""""""""""""""""""""""""""""""""""""""""""""""""""""""

    logits = tf.layers.dense(
        inputs=dense1,
        units=87
    )

    predictions = {
        "classes": tf.argmax(
            input=logits,
            axis=1
        ),
        "probabilities": tf.nn.softmax(
            logits=logits,
            name="softmax_tensor"
        )
    }

    if mode == tf.estimator.ModeKeys.PREDICT:

        return tf.estimator.EstimatorSpec(
            mode=mode,
            predictions=predictions
        )

    loss = tf.losses.sparse_softmax_cross_entropy(
        labels=labels,
        logits=logits
    )

    if mode == tf.estimator.ModeKeys.EVAL:

        eval_metric_ops = {
            "accuracy": tf.metrics.accuracy(
                labels=labels,
                predictions=predictions["classes"]
            )
        }

        return tf.estimator.EstimatorSpec(
            mode=mode,
            loss=loss,
            eval_metric_ops=eval_metric_ops
        )

    if mode == tf.estimator.ModeKeys.TRAIN:

        with
```

```
tf.control_dependencies(tf.get_collection(tf.GraphKeys.UPDATE_OPS)):

            train_op = tf.train.AdamOptimizer().minimize(
                loss=loss,
                global_step=tf.train.get_global_step()
            )

        return tf.estimator.EstimatorSpec(
            mode=mode,
            loss=loss,
            train_op=train_op
        )

def main(unused_argv):

    tag2vec = gensim.models.word2vec.Word2Vec.load("tag2vec.model").wv
    label2vec = gensim.models.word2vec.Word2Vec.load("label2vec.model").wv
    word2vec =
gensim.models.KeyedVectors.load_word2vec_format("google_word2vec_model.bin
", binary=True)

    Item = collections.namedtuple("Item", ("index", "word", "tag",
"label", "parent", "children"))
    train_sentences = [[Item(0, "", "", "", -1, [])]]
    eval_sentences = [[Item(0, "", "", "", -1, [])]]

    with open("train.conll") as file:

        for line in file:

            line = line.split()

            if line:

                index, word, _, tag, _, _, parent, label, _, _ = line
                train_sentences[-1].append(Item(index, word, tag, label,
parent, []))

            else:

                train_sentences.append([Item(0, "", "", "", -1, [])])

    with open("eval.conll") as file:

        for line in file:

            line = line.split()

            if line:

                index, word, _, tag, _, _, parent, label, _, _ = line
                eval_sentences[-1].append(Item(index, word, tag, label,
parent, []))
```

```
            else:

                eval_sentences.append([Item(0, "", "", "", -1, [])])

    label_id = {}

    for label in label2vec.vocab:

        label_id[label] = len(label_id)

    def embed(sentences):

        data = []
        labels = []

        for buffer in sentences:

            if len(buffer) < 2: continue

            stack = []

            stack.append(buffer.pop(0))
            stack.append(buffer.pop(0))

            while True:

                concat = []

                def try_word2vec(word):

                    return word2vec[word] if word in word2vec else
np.zeros(300)

                def try_tag2vec(tag):

                    return tag2vec[tag] if tag in tag2vec else
np.zeros(100)

                def try_label2vec(label):

                    return label2vec[label] if label in label2vec else
np.zeros(100)


"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                Sw contains 18 elements

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                concat.append(try_word2vec(stack[-1].word) if len(stack) >
0 else np.zeros(300))
                concat.append(try_word2vec(stack[-2].word) if len(stack) >
1 else np.zeros(300))
                concat.append(try_word2vec(stack[-3].word) if len(stack) >
2 else np.zeros(300))
```

```
                concat.append(try_word2vec(buffer[0].word) if len(buffer)
> 0 else np.zeros(300))
                concat.append(try_word2vec(buffer[1].word) if len(buffer)
> 1 else np.zeros(300))
                concat.append(try_word2vec(buffer[2].word) if len(buffer)
> 2 else np.zeros(300))

                concat.append(try_word2vec(stack[-1].children[0].word) if
len(stack) > 0 and len(stack[-1].children) > 0 else np.zeros(300))
                concat.append(try_word2vec(stack[-1].children[1].word) if
len(stack) > 0 and len(stack[-1].children) > 1 else np.zeros(300))
                concat.append(try_word2vec(stack[-1].children[-1].word) if
len(stack) > 0 and len(stack[-1].children) > 0 else np.zeros(300))
                concat.append(try_word2vec(stack[-1].children[-2].word) if
len(stack) > 0 and len(stack[-1].children) > 1 else np.zeros(300))

                concat.append(try_word2vec(stack[-2].children[0].word) if
len(stack) > 1 and len(stack[-2].children) > 0 else np.zeros(300))
                concat.append(try_word2vec(stack[-2].children[1].word) if
len(stack) > 1 and len(stack[-2].children) > 1 else np.zeros(300))
                concat.append(try_word2vec(stack[-2].children[-1].word) if
len(stack) > 1 and len(stack[-2].children) > 0 else np.zeros(300))
                concat.append(try_word2vec(stack[-2].children[-2].word) if
len(stack) > 1 and len(stack[-2].children) > 1 else np.zeros(300))


concat.append(try_word2vec(stack[-1].children[0].children[0].word) if
len(stack) > 0 and len(stack[-1].children) > 0 and
len(stack[-1].children[0].children) > 0 else np.zeros(300))

concat.append(try_word2vec(stack[-1].children[-1].children[-1].word) if
len(stack) > 0 and len(stack[-1].children) > 0 and
len(stack[-1].children[-1].children) > 0 else np.zeros(300))


concat.append(try_word2vec(stack[-2].children[0].children[0].word) if
len(stack) > 1 and len(stack[-2].children) > 0 and
len(stack[-2].children[0].children) > 0 else np.zeros(300))

concat.append(try_word2vec(stack[-2].children[-1].children[-1].word) if
len(stack) > 1 and len(stack[-2].children) > 0 and
len(stack[-2].children[-1].children) > 0 else np.zeros(300))


"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                St contains 18 elements

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                concat.append(try_tag2vec(stack[-1].tag) if len(stack) > 0
else np.zeros(100))
                concat.append(try_tag2vec(stack[-2].tag) if len(stack) > 1
else np.zeros(100))
                concat.append(try_tag2vec(stack[-3].tag) if len(stack) > 2
else np.zeros(100))

                concat.append(try_tag2vec(buffer[0].tag) if len(buffer) >
0 else np.zeros(100))
                concat.append(try_tag2vec(buffer[1].tag) if len(buffer) >
1 else np.zeros(100))
                concat.append(try_tag2vec(buffer[2].tag) if len(buffer) >
2 else np.zeros(100))

                concat.append(try_tag2vec(stack[-1].children[0].tag) if
len(stack) > 0 and len(stack[-1].children) > 0 else np.zeros(100))
                concat.append(try_tag2vec(stack[-1].children[1].tag) if
len(stack) > 0 and len(stack[-1].children) > 1 else np.zeros(100))
                concat.append(try_tag2vec(stack[-1].children[-1].tag) if
len(stack) > 0 and len(stack[-1].children) > 0 else np.zeros(100))
                concat.append(try_tag2vec(stack[-1].children[-2].tag) if
len(stack) > 0 and len(stack[-1].children) > 1 else np.zeros(100))

                concat.append(try_tag2vec(stack[-2].children[0].tag) if
len(stack) > 1 and len(stack[-2].children) > 0 else np.zeros(100))
                concat.append(try_tag2vec(stack[-2].children[1].tag) if
len(stack) > 1 and len(stack[-2].children) > 1 else np.zeros(100))
                concat.append(try_tag2vec(stack[-2].children[-1].tag) if
len(stack) > 1 and len(stack[-2].children) > 0 else np.zeros(100))
                concat.append(try_tag2vec(stack[-2].children[-2].tag) if
len(stack) > 1 and len(stack[-2].children) > 1 else np.zeros(100))


concat.append(try_tag2vec(stack[-1].children[0].children[0].tag) if
len(stack) > 0 and len(stack[-1].children) > 0 and
len(stack[-1].children[0].children) > 0 else np.zeros(100))

concat.append(try_tag2vec(stack[-1].children[-1].children[-1].tag) if
len(stack) > 0 and len(stack[-1].children) > 0 and
len(stack[-1].children[-1].children) > 0 else np.zeros(100))


concat.append(try_tag2vec(stack[-2].children[0].children[0].tag) if
len(stack) > 1 and len(stack[-2].children) > 0 and
len(stack[-2].children[0].children) > 0 else np.zeros(100))

concat.append(try_tag2vec(stack[-2].children[-1].children[-1].tag) if
len(stack) > 1 and len(stack[-2].children) > 0 and
len(stack[-2].children[-1].children) > 0 else np.zeros(100))


""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                Sl contains 12 elements

""""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                concat.append(try_label2vec(stack[-1].children[0].label)
if len(stack) > 0 and len(stack[-1].children) > 0 else np.zeros(100))
                concat.append(try_label2vec(stack[-1].children[1].label)
if len(stack) > 0 and len(stack[-1].children) > 1 else np.zeros(100))
```

```
                concat.append(try_label2vec(stack[-1].children[-1].label)
if len(stack) > 0 and len(stack[-1].children) > 0 else np.zeros(100))
                concat.append(try_label2vec(stack[-1].children[-2].label)
if len(stack) > 0 and len(stack[-1].children) > 1 else np.zeros(100))

                concat.append(try_label2vec(stack[-2].children[0].label)
if len(stack) > 1 and len(stack[-2].children) > 0 else np.zeros(100))
                concat.append(try_label2vec(stack[-2].children[1].label)
if len(stack) > 1 and len(stack[-2].children) > 1 else np.zeros(100))
                concat.append(try_label2vec(stack[-2].children[-1].label)
if len(stack) > 1 and len(stack[-2].children) > 0 else np.zeros(100))
                concat.append(try_label2vec(stack[-2].children[-2].label)
if len(stack) > 1 and len(stack[-2].children) > 1 else np.zeros(100))


concat.append(try_label2vec(stack[-1].children[0].children[0].label) if
len(stack) > 0 and len(stack[-1].children) > 0 and
len(stack[-1].children[0].children) > 0 else np.zeros(100))

concat.append(try_label2vec(stack[-1].children[-1].children[-1].label) if
len(stack) > 0 and len(stack[-1].children) > 0 and
len(stack[-1].children[-1].children) > 0 else np.zeros(100))


concat.append(try_label2vec(stack[-2].children[0].children[0].label) if
len(stack) > 1 and len(stack[-2].children) > 0 and
len(stack[-2].children[0].children) > 0 else np.zeros(100))

concat.append(try_label2vec(stack[-2].children[-1].children[-1].label) if
len(stack) > 1 and len(stack[-2].children) > 0 and
len(stack[-2].children[-1].children) > 0 else np.zeros(100))

                data.append(np.concatenate(concat))

                if len(stack) >= 2 and stack[-1].parent ==
stack[-2].index:


"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                    RIGHT-ARC (stack[-2] => stack[-1])

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

                    labels.append(label_id[stack[-1].label] +
len(label_id) * 0 + 1)
                    stack[-2].children.append(stack.pop(-1))

                elif len(stack) >= 2 and stack[-2].parent ==
stack[-1].index:


"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                    LEFT-ARC (stack[-2] <= stack[-1])
```

```
"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

                    labels.append(label_id[stack[-1].label] +
len(label_id) * 1 + 1)
                    stack[-1].children.append(stack.pop(-2))

                else:


"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""
                    SHIFT

"""""""""""""""""""""""""""""""""""""""""""""""""""""""""""

                    labels.append(0)

                    if not buffer: break

                    stack.append(buffer.pop(0))

        return np.array(data), np.array(labels)

    train_data, train_labels = embed(train_sentences)
    eval_data, eval_labels = embed(eval_sentences)

    print(train_data.shape)
    print(eval_data.shape)

    run_config = tf.estimator.RunConfig().replace(
        session_config=tf.ConfigProto(device_count={'GPU': 1}))

    dependenc_estimator = tf.estimator.Estimator(
        model_fn=dependency_parser_model_fn,
        model_dir=args.model,
        config=run_config
    )

    if args.train:

        train_input_fn = tf.estimator.inputs.numpy_input_fn(
            x={"x": train_data},
            y=train_labels,
            batch_size=args.batch,
            num_epochs=None,
            shuffle=True
        )

        logging_hook = tf.train.LoggingTensorHook(
            tensors={
                "probabilities": "softmax_tensor"
            },
            every_n_iter=100
        )
        dependenc_estimator.train(
            input_fn=train_input_fn,
            steps=args.steps,
            hooks=[logging_hook]
        )

    if args.eval:

        eval_input_fn = tf.estimator.inputs.numpy_input_fn(
            x={"x": eval_data},
            y=eval_labels,
            num_epochs=1,
            shuffle=False
        )

        eval_results = dependenc_estimator.evaluate(
            input_fn=eval_input_fn
        )

        print(eval_results)


if __name__ == "__main__":
    tf.app.run()
```