Основы Keras и PyTorch. BiLSTM+CRF бейзлайн.

Консультация

Перминов Андрей Игоревич

12 ноября 2021

Наиболее популярные фреймворки ML

Keras

- предлагает последовательные и простые API-интерфейсы
- сводит к минимуму количество действий пользователя, необходимых для типичных случаев использования
- содержит обширную документацию и руководства для разработчиков
- модели строятся по принципу конструктора из готовых блоков
- быстрый запуск обучения

PyTorch

- позволяет строить вычислительные динамические графы
- больше действий для экспериментов с моделями
- более гибкая настройка моделей
- необходимо самостоятельно описывать процедуру обучения
- более сложный, но более мощный инструмент

Keras. Пример полносвязной сети

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(12, input_dim=8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='softmax'))

model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

x_train, y_train = ...

model.fit(x_train, y_train, epochs=150, batch_size=10)
y_test = model.predict(x_test)
```

PyTorch. Пример полносвязной сети

```
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torch.autograd import Variable


class Network(nn.Module):
    def __init__(self):
        super(Network, self).__init__()
        self.l1 = nn.Linear(8, 12)
        self.relu1 = nn.ReLU()
        self.l2 = nn.Linear(12, 8)
        self.relu2 = nn.ReLU()
        self.l3 = nn.Linear(8, 1)

    def forward(self, x):
        x = self.l1(x)
        x = self.relu1(x)
        x = self.l2(x)
        x = self.relu2(x)
        x = self.l3(x)
        return F.log_softmax(x)
```

PyTorch. Пример полносвязной сети

```
model = Network()
optimizer = optim.Adam(model.parameters())
loss_func = nn.CrossEntropyLoss()

x_train, y_train = ...

batch_size = 10
epochs = 150

for e in range(epochs):
    for i in range(0, x_train.shape[0], batch_size):
        x_var = Variable(x_train[i:i + batch_size])
        y_var = Variable(y_train[i:i + batch_size])

        optimizer.zero_grad()
        net_out = model(x_var)

        loss = loss_func(net_out, y_var)
        loss.backward()
        optimizer.step()

    print('Epoch: {} - Loss: {:.6f}'.format(e, loss.data[0]))

net_out = model(x_test)
```

Keras. BiLSTM для IMDB

```
from tensorflow import keras
from tensorflow.keras import layers

max_features = 20000
maxlen = 200

inputs = keras.Input(shape=(None,), dtype='int32')
x = layers.Embedding(max_features, 128)(inputs)
x = layers.Bidirectional(layers.LSTM(64, return_sequences=True))(x)
x = layers.Bidirectional(layers.LSTM(64))(x)
outputs = layers.Dense(1, activation='sigmoid')(x)
model = keras.Model(inputs, outputs)
model.summary()

(x_train, y_train), (x_val, y_val) = keras.datasets.imdb.load_data(num_words=max_features)

x_train = keras.preprocessing.sequence.pad_sequences(x_train, maxlen=maxlen)
x_val = keras.preprocessing.sequence.pad_sequences(x_val, maxlen=maxlen)

model.compile('adam', 'binary_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=32, epochs=2, validation_data=(x_val, y_val))
```

Baseline решение для NERC

Работает в несколько этапов:

- токенизация текстов новостей
- BIO кодирование
- загрузка word2vec эмбеддингов
- создание BiLSTM + CRF модели
- разрешение коллизий разметки
- обучение модели
- предсказание модели
- BIO декодирование

Токенизация

Поскольку для задачи NERC необходимо знать индексы сущностей в исходном тексте, обычная токенизация на строки не подходит. Необходимо токенизировать на span'ы – кортежи из индексов начала и конца токена. Наиболее популярные варианты – NLTK и регулярные выражения:

```
def nltk_tokenize(self, text: str) -> List[Tuple[int, int]]:
    tokenizer = nltk.tokenize.TreebankWordTokenizer()
    return  tokenizer.span_tokenize(text)
```

```
def regexp_tokenize(self, text: str) -> List[Tuple[int, int]]:
    spans = [(match.start(), match.end()) for match in self.tokenize_regexp.finditer(text)]
    return spans
```

BIO кодировка

По имеющимся сущностям формируется словарь индекс – метка (B для start и O для остальных значений span’a). Затем по полученным span’ам формируется BIO разметка всего текста.

```
entities_dict = {}

for start, end, name in entities:
    entities_dict[start] = f'B-{name}'
    for i in range(start + 1, end):
        entities_dict[i] = f‘I-{name}'

spans = tokenize(text)
bio = []

for start, end in spans:
    label = entities_dict.get(start, 'O')
    bio.append(self.bio2index[label])
```

BIO кодировка. Пример

Текст:
[18 апреля 2016 года|DATE]

["Арт Пикчерс Студия"|ORGANIZATION] в лице продюсера [Дмитрия Рудовского|PERSON] представила самый секретный проект компании ["Притяжение"|ORGANIZATION]. Главную роль в картине играет [Олег Меньшиков|PERSON]. Режиссером выступает [Федор Бондарчук|PERSON].

Сущности:
(0, 19, 'DATE'), (21, 41, 'ORGANIZATION'), (59, 77, 'PERSON'), (122, 134, 'ORGANIZATION'), (166, 180, 'PERSON'), (203, 218, 'PERSON')

Результат токенизации (в виде строк):
['18', 'апреля', '2016', 'года', '"', 'Арт', 'Пикчерс', 'Студия', '"', 'в', 'лице', 'продюсера', 'Дмитрия', 'Рудовского', 'представила', 'самый', 'секретный', 'проект', 'компании', '"', 'Притяжение', '"', '.', 'Главную', 'роль', 'в', 'картине', 'играет', 'Олег', 'Меньшиков', '.', 'Режиссером', 'выступает', 'Федор', 'Бондарчук', '.']

BIO кодировка. Пример

Результат токенизации (в виде строк):
['18', 'апреля', '2016', 'года', '"', 'Арт', 'Пикчерс', 'Студия', '"', 'в', 'лице', 'продюсера', 'Дмитрия', 'Рудовского', 'представила', 'самый', 'секретный', 'проект', 'компании', '"', 'Притяжение', '"', '.', 'Главную', 'роль', 'в', 'картине', 'играет', 'Олег', 'Меньшиков', '.', 'Режиссером', 'выступает', 'Федор', 'Бондарчук', '.']

Результат токенизации (в виде span'ов):
[(0, 2), (3, 9), (10, 14), (15, 19), (21, 22), (22, 25), (26, 33), (34, 40), (40, 41), (42, 43), (44, 48), (49, 58), (59, 66), (67, 77), (78, 89), (90, 95), (96, 105), (106, 112), (113, 121), (122, 123), (123, 133), (133, 134), (134, 135), (136, 143), (144, 148), (149, 150), (151, 158), (159, 165), (166, 170), (171, 180), (180, 181), (182, 192), (193, 202), (203, 208), (209, 218), (218, 219)]

Результат BIO кодирования:
['B-DATE', 'I-DATE', 'I-DATE', 'I-DATE', 'B-ORGANIZATION', 'I-ORGANIZATION', 'I-ORGANIZATION', 'I-ORGANIZATION', 'I-ORGANIZATION', 'O', 'O', 'O', 'B-PERSON', 'I-PERSON', 'O', 'O', 'O', 'O', 'O', 'B-ORGANIZATION', 'I-ORGANIZATION', 'I-ORGANIZATION', 'O', 'O', 'O', 'O', 'O', 'O', 'B-PERSON', 'I-PERSON', 'O', 'O', 'O', 'B-PERSON', 'I-PERSON', 'O']

Загрузка Word2Vec эмбеддингов

В качестве векторного представления слов используется эмбеддинг с 100-мерными векторами и окном 5 (w2v_size100_window5.txt).

```
import gensim

word_vectors = gensim.models.KeyedVectors.load_word2vec_format('/embeddings/w2v_size100_window5.txt')
embedding_dim = self.word_vectors.vector_size
```

Создание BiLSTM + CRF модели

```
START_TAG = "<START>"
STOP_TAG = "<STOP>"

class BiLSTM_CRF(nn.Module):
    def __init__(self, tag_to_ix: dict, lstm_size: int, embedding_path: str):
        super(BiLSTM_CRF, self).__init__()
        self.hidden_dim = lstm_size

        self.tag_to_ix = {tag: index for tag, index in tag_to_ix.items()}
        self.tag_to_ix[START_TAG] = len(self.tag_to_ix)
        self.tag_to_ix[STOP_TAG] = len(self.tag_to_ix)

        self.tagset_size = len(self.tag_to_ix)

        self.word_vectors = gensim.models.KeyedVectors.load_word2vec_format(embedding_path)
        self.embedding_dim = self.word_vectors.vector_size

        self.lstm = nn.LSTM(self.embedding_dim, lstm_size // 2, num_layers=1, bidirectional=True)
        self.hidden2tag = nn.Linear(lstm_size, self.tagset_size)

        self.transitions = nn.Parameter(torch.randn(self.tagset_size, self.tagset_size))
        self.transitions.data[self.tag_to_ix[START_TAG], :] = -10000
        self.transitions.data[:, self.tag_to_ix[STOP_TAG]] = -10000
```

Создание BiLSTM + CRF модели

```
def _get_lstm_features(self, sentence):
    self.hidden = self.init_hidden()
    embeds = self._get_embedded(sentence).view(len(sentence), 1, -1)
    lstm_out, self.hidden = self.lstm(embeds, self.hidden)
    lstm_out = lstm_out.view(len(sentence), self.hidden_dim)
    lstm_feats = self.hidden2tag(lstm_out)
    return lstm_feats

def forward(self, sentence):
    lstm_feats = self._get_lstm_features(sentence)
    score, tag_seq = self._viterbi_decode(lstm_feats)
    return score, tag_seq
```

Разрешение коллизий разметки

Поскольку один текст может быть размечен от 1 до 3 людьми, необходимо определить, какие именно сущности отдавать модели. Можно использовать следующие подходы:

- пересечение всех разметок
- объединение всех разметок
- голосование
- все разметки по отдельности, возможно, модель самостоятельно справится с шумом
- более хитрые схемы с учётом особенностей домена

Обучение модели

Модель обучается в режиме стохастического градиентного спуска.
По истечении 25 минут обучение принудительно прекращается.

```
def train(self, train_corpus: List[Tuple[str, Dict[str, Set[Tuple[int, int, str]]]]]):
    start_time = time.perf_counter()
    train_tokens, train_bio = [], []

    for text, user2label in train_corpus:
        for entities in user2label.values():
            spans, bio = self.bio_encoder.encode(text, entities)
            train_tokens.append([text[start:end] for start, end in spans])
            train_bio.append(bio)

    indexes = [i for i in range(len(train_tokens))]
    random.shuffle(indexes)
```

Обучение модели

Модель обучается в режиме стохастического градиентного спуска. По истечении 25 минут обучение принудительно прекращается.

```
for epoch in range(self.max_epochs):
    for i in range(len(train_tokens)):
        tokens = train_tokens[indexes[i]]
        bio = train_bio[indexes[i]]

        self.model.zero_grad()
        targets = torch.tensor(bio, dtype=torch.long)
        loss = self.model.neg_log_likelihood(tokens, targets)
        loss.backward()
        self.optimizer.step()

        delta = time.perf_counter() - start_time

        if delta >= self.max_train_time:
            return
```

Предсказание модели

Каждый текст токенизируется, модель вычисляет индексы BIO разметки, которая затем декодируется обратно в span'ы.

```
def predict_text(self, text: str) -> Set[Tuple[int, int, str]]:
    spans = self.bio_encoder.tokenize(text)
    tokens = [text[start:end] for start, end in spans]

    predicted = self.model(tokens)[1]
    decoded = self.bio_encoder.decode(spans, predicted)

    return decoded

def predict(self, news: List[str]) -> List[Set[Tuple[int, int, str]]]:
    with torch.no_grad():
        entities = [self.predict_text(text) for text in news]

    return entities
```

BIO декодирование

Ищется вхождение стартовой (B) метки. Сущность формируется пока следующие метки являются внутренними (I) и добавляется во множество.

```
def decode(self, spans: List[Tuple[int, int]], bio: List[int]) -> Set[Tuple[int, int, str]]:
    entities = set()
    i = 0

    while i < len(bio):
        while i < len(bio) and not self.index2bio[bio[i]].startswith(BEGIN):
            i += 1

        if i == len(bio):
            break

        start, end = spans[i]
        label = self.bio2name[self.index2bio[bio[i]]]
        i += 1

        while i < len(bio) and self.index2bio[bio[i]].startswith(INNER):
            end = spans[i][1]
            i += 1

        entities.add((start, end, label))

    return entities
```

Полезные ссылки

- https://keras.io/
- https://pytorch.org/
- https://keras.io/examples/
- https://pytorch.org/tutorials/beginner/pytorch_with_examples.html
- https://pytorch.org/tutorials/beginner/nlp/advanced_tutorial.html
- https://towardsdatascience.com/named-entity-recognition-and-classification-with-scikit-learn-f05372f07ba2

Вопросы?