Лабораторная работа №8.

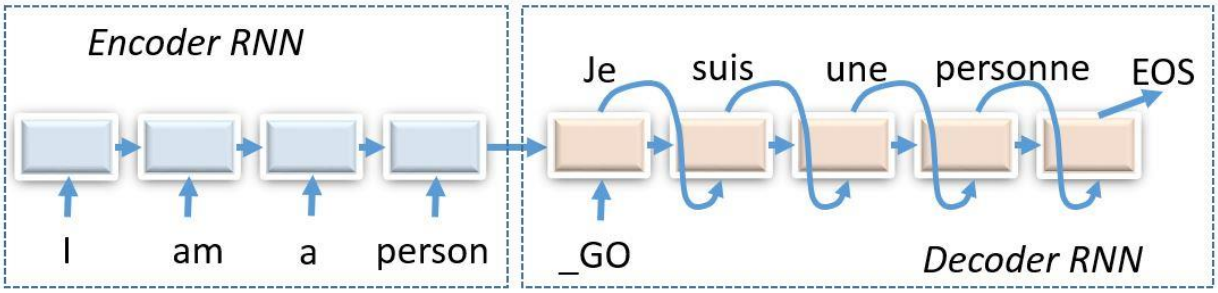
Обучение с подкреплением для seq2seq

На этот раз мы решим задачу транскрипции слов на иврите на английском языке, также известную как g2p (grapheme2phoneme) слово (последовательность букв на исходном языке) -> перевод (последовательность букв на языке перевода). В отличие от того, что делают большинство методов глубокого обучения, мы не только будем обучать модель для максимизации вероятности правильного перевода, но и использовать обучение с подкреплением, чтобы фактически научить модель переводить с минимальным количеством ошибок.

О задаче.

Одним из примечательных свойств иврита является то, что это согласный язык. То есть в письменном языке нет гласных. Можно представить гласные с диакритическими знаками над согласными, но никто не ожидает, что люди будут делать это в повседневной жизни.

Таким образом, некоторые символы иврита будут соответствовать нескольким английским буквам, а другие — ни одной, поэтому мы должны использовать архитектуру кодировщик-декодер, чтобы понять это.



Архитектуры кодер-декодер предназначены для преобразования чего угодно во что угодно, в том числе:

- Системы машинного перевода и разговорного диалога
- Подписи к изображениям и image2latex (сверточный кодировщик, рекуррентный декодер)
- Генерация изображений по подписям (рекуррентный кодировщик, сверточный декодер)
- Grapheme2phoneme - конвертировать слова в транскрипты

Мы выбрали упрощенный машинный перевод с иврита на английский для слов и коротких фраз (на уровне символов), так как его можно относительно быстро обучить даже без кластера графического процессора.

1. Предварительные действия

В работе используется файлы:

- «main_dataset.txt», который требуется скачать по ссылке: https://github.com/sshkhr/Practical_RL/blob/master/week8_scst/main_dataset.txt
- «basic_model_tf.py», который требуется скачать по ссылке: https://github.com/sshkhr/Practical_RL/blob/master/week8_scst/basic_model_tf.py
- «», который требуется скачать по ссылке: https://github.com/sshkhr/Practical_RL/blob/master/week8_scst/basic_model_torch.py

```
import sys, os, time

# Этот код создает виртуальный дисплей для рисования игровых изображений.
# Это не будет иметь никакого эффекта, если на вашей машине есть монитор.
if type(os.environ.get("DISPLAY")) is not str or
len(os.environ.get("DISPLAY")) == 0:
    os.environ['DISPLAY'] = ':1'


EASY_MODE = True         #Если True, переводит только фразы короче 20
символов (намного проще).
                         #Полезно для начального кодирования.
                         #Если false, работает со всеми фразами

MODE = "he-to-en"                                #способ перевода. Или "he-
to-en" или "en-to-he"
MAX_OUTPUT_LENGTH = 50 if not EASY_MODE else 20  #максимальная длина
_generated_ вывода, не влияет на обучение
REPORT_FREQ       = 100                          #насколько часто оценивать
меру проверки
```

2. Препроцессинг

Мы будем хранить набор данных в виде словаря {слово1:[перевод1,перевод2,...], слово2:[...],...}. В основном это связано с тем, что многие слова имеют несколько правильных переводов. Эта функциональность уже реализована.

```
import numpy as np
from collections import defaultdict
word_to_translation = defaultdict(list)  # наш словарь

bos = '_'
eos = ';'

with open("main_dataset.txt", encoding='utf8') as fin:
    for line in fin:

        en, he = line[:-1].lower().replace(bos, ' ').replace(eos,
                                                             ' ').split('\t')
        word, trans = (he, en) if MODE == 'he-to-en' else (en, he)

        if len(word) < 3:
            continue
        if EASY_MODE:
            if max(len(word), len(trans)) > 20:
                continue

        word_to_translation[word].append(trans)

print("size = ", len(word_to_translation))
```

Вывод:

size = 130114

```
# получить все уникальные строки на исходном языке
all_words = np.array(list(word_to_translation.keys()))
# получить все уникальные строки на языке перевода
all_translations = np.array(
    [ts for all_ts in word_to_translation.values() for ts in all_ts])
```

Разделить набор данных.

Мы удерживаем 10% всех слов, которые будут использоваться для проверки.

```
from sklearn.model_selection import train_test_split
train_words, test_words = train_test_split(
    all_words, test_size=0.1, random_state=42)
```

Построение словарей

Теперь нам нужно создать словари, которые сопоставляют строки с идентификаторами токенов и наоборот. Нам понадобятся эти записи, когда мы будем вводить обучающие данные в модель или преобразовывать выходные матрицы в английские слова.

```
from voc import Vocab
inp_voc = Vocab.from_lines(''.join(all_words), bos=bos, eos=eos, sep='')
out_voc = Vocab.from_lines(''.join(all_translations), bos=bos, eos=eos,
sep='')
```

```
# перевод строк в идентификаторы и обратно.
batch_lines = all_words[:5]
batch_ids = inp_voc.to_matrix(batch_lines)
batch_lines_restored = inp_voc.to_lines(batch_ids)

print("lines")
print(batch_lines)
print("\nwords to ids (0 = bos, 1 = eos):")
print(batch_ids)
print("\nback to words")
print(batch_lines_restored)
```

Вывод:

lines
['משתמש:צלף/!' 'סימן קריאה' 'תבנית:!!' '$9.99' "תבנית:'"]

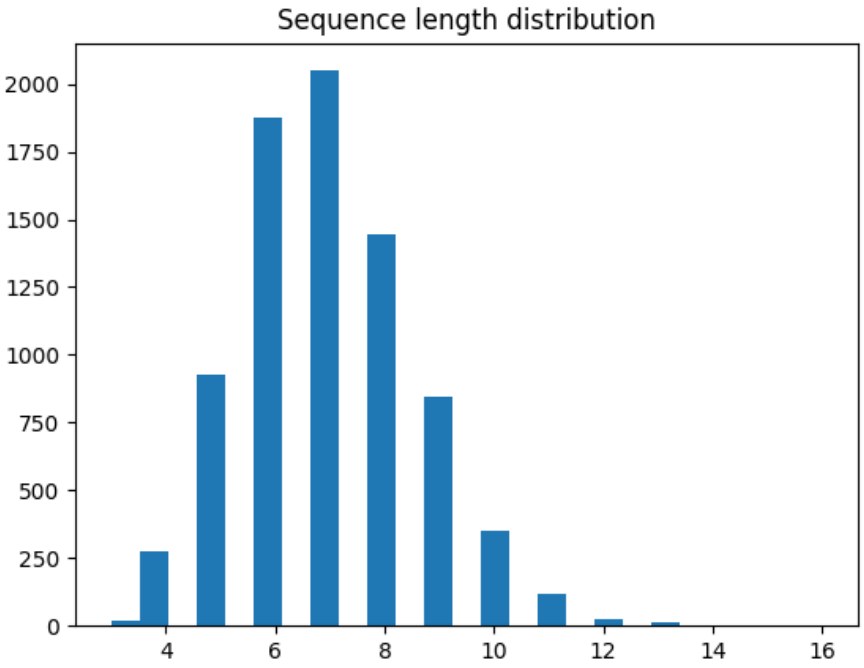
words to ids (0 = bos, 1 = eos):
[[0 127 138 139 127 138 27 135 125 132 16 3 1]
[0 130 122 127 128 2 136 137 122 113 117 1 1]
[0 139 114 129 122 139 27 3 3 1 1 1 1]
[0 6 26 15 26 26 1 1 1 1 1 1 1]
[0 139 114 129 122 139 27 8 1 1 1 1 1]]

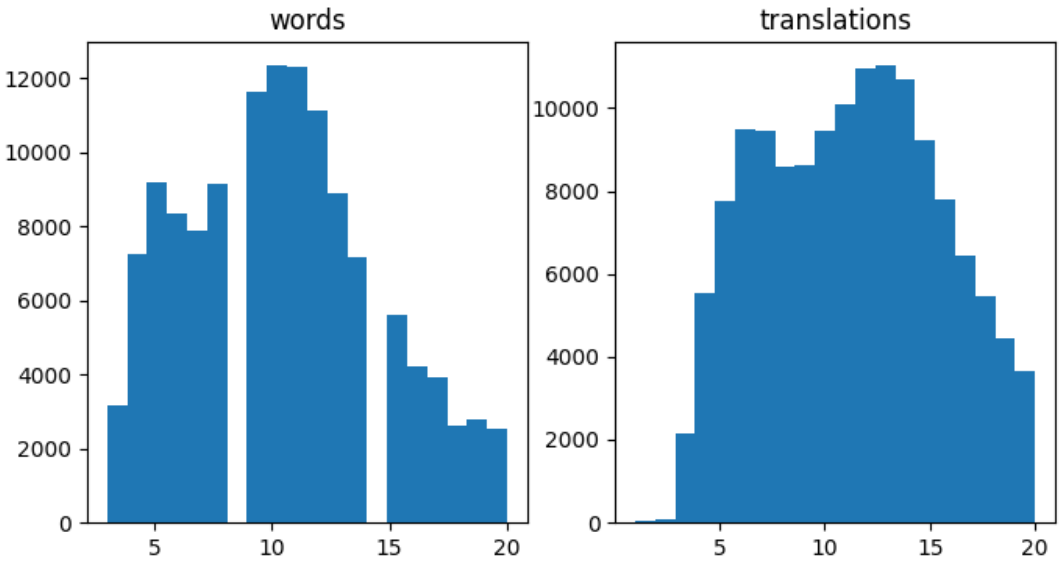
back to words
['משתמש:צלף/!', 'סימן קריאה', 'תבנית:!!', '$9.99', "תבנית:'"]

Нарисуйте распределение длины слова/перевода, чтобы оценить объем задачи.

```
import matplotlib.pyplot as plt
plt.figure(figsize=[8, 4])
plt.subplot(1, 2, 1)
plt.title("words")
plt.hist(list(map(len, all_words)), bins=20)

plt.subplot(1, 2, 2)
plt.title('translations')
plt.hist(list(map(len, all_translations)), bins=20)
plt.show()
```

Вывод:





3. Развертывание кодировщика-декодера

Наша архитектура состоит из двух основных блоков:

- Кодер считывает слова посимвольно и выводит кодовый вектор (обычно это функция последнего состояния RNN).
- Декодер берет этот кодовый вектор и производит перевод символ за символом.

Затем он передается в модель, которая следует этому простому интерфейсу:

- model.symbolic_translate(inp, **flags) -> out, logp — принимает символическую матрицу слов на иврите int32, создает выходные токены, выбранные из модели, и выводит логарифмические вероятности для всех возможных токенов на каждом тике.

- model.symbolic_score(inp, out, **flags) -> logp - принимает символьные матрицы int32 слов на иврите и их английские переводы. Вычисляет логарифмические вероятности всех возможных английских символов с учетом английских префиксов и еврейского слова.
- model.weights - веса всех слоев модели [список переменных]

С помощью этих двух методов вы можете реализовать все виды прогнозирования и обучения.

```
import tensorflow.compat.v1 as tf
tf.disable_v2_behavior()

tf.compat.v1.reset_default_graph()
s = tf.compat.v1.InteractiveSession()

from basic_model_tf import BasicTranslationModel
model = BasicTranslationModel('model', inp_voc, out_voc,
                              emb_size=64, hid_size=128)

s.run(tf.compat.v1.global_variables_initializer())
```

```
# Поэкспериментируйте с symbolic_translate и symbolic_score
inp = tf.placeholder_with_default(np.random.randint(
    0, 10, [3, 5], dtype='int32'), [None, None])
out = tf.placeholder_with_default(np.random.randint(
    0, 10, [3, 5], dtype='int32'), [None, None])

# перевести ввод (с необученной моделью)
sampled_out, logp = model.symbolic_translate(inp, greedy=False)
print("\nSymbolic_translate output:\n", sampled_out, logp)
print("\nSample translations:\n", s.run(sampled_out))
```

Вывод:

Symbolic_translate output:
Tensor("transpose_1:0", shape=(?, ?), dtype=int32) Tensor("LogSoftmax:0", shape=(?, ?, 283), dtype=float32)

Sample translations:
[[0 109 66 82 78 119 169 92 16 32 216]
[0 118 94 194 124 281 99 110 10 7 66]
[0 92 247 154 197 138 133 96 28 56 34]]

```
# оценка logp(out | inp) с необученным вводом
logp = model.symbolic_score(inp, out)
print("\nSymbolic_score output:\n", logp)
print("\nLog-probabilities (clipped):\n", s.run(logp)[:, :2, :5])
```

Вывод:

Symbolic_score output:
Tensor("LogSoftmax_1:0", shape=(?, ?, 283), dtype=float32)

Log-probabilities (clipped):
[[[0. -69.07755 -69.07755 -69.07755 -69.07755]
[-5.6537566 -5.651446 -5.6586833 -5.6411147 -5.649842]]

[[0. -69.07755 -69.07755 -69.07755 -69.07755]
[-5.660492 -5.654201 -5.6510887 -5.631667 -5.6455073]]

[[0.   -69.07755   -69.07755   -69.07755   -69.07755]
 [-5.661183   -5.647964   -5.653283   -5.626666   -5.647372]]]

Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
# Секция для самостоятельных заданий
input_sequence = tf.placeholder('int32', [None, None])
greedy_translations, logp = <ВАШ КОД: создавайте символические переводы с помощью greedy = True>


def translate(lines):
    """
    Вам дан список входных строк.
     Заставьте свою нейронную сеть перевести их.
     :return: список строк вывода
    """
    # Преобразуйте линии в матрицу индексов
    lines_ix = <ВАШ КОД>

    # Вычислить переводы в виде индексов
    trans_ix = s.run(greedy_translations, { <ВАШ КОД: feed_dict> })

    # Преобразование переводов обратно в строки
    return out_voc.to_lines(trans_ix)
```

```
print("Sample inputs:", all_words[:3])
print("Dummy translations:", translate(all_words[:3]))

assert isinstance(greedy_translations,
                  tf.Tensor) and greedy_translations.dtype.is_integer, "trans должен быть тензором целых чисел (идентификаторов токенов)"
assert translate(all_words[:3]) == translate(
    all_words[:3]), "убедитесь, что перевод является детерминированным (используйте greedy=True и отключите все шумовые слои)"
assert type(translate(all_words[:3])) is list and (type(translate(all_words[:1])[0]) is str or type(
    translate(all_words[:1])[0]) is unicode), "translate(lines) должен возвращать последовательность строк!"
print("Tests passed!")
```

Вывод:

Sample inputs: ['משתמש:צלף/!' 'קריאה סימן' 'תבנית:!!']
Dummy translations: ['צשẃżצ子子ù子ùż子つつつつつつつつ陳ոոկ0ó0', 'ηשσσسسđää榎ä榎կõõ777vvגגòと&熊', 'שñỏñťηと&ỏñ×%ỹṣяяćć仙ペペペ×%ππ']
Tests passed!

Скоринговая функция

LogLikelihood — плохая оценка производительности модели.

- Если мы предсказываем нулевую вероятность один раз, это не должно разрушить всю модель.
- Достаточно выучить только один перевод, если правильных несколько.
- Важно то, сколько ошибок сделает модель при переводе!

Поэтому будем использовать минимальное расстояние Левенштейна. Оно измеряет, сколько символов нам нужно добавить/удалить/заменить из перевода модели, чтобы сделать его идеальным. В качестве альтернативы можно использовать BLEU/RougeL на уровне персонажа или другие подобные показатели.

Загвоздка здесь в том, что расстояние Левенштейна не дифференцируемо: оно даже не является непрерывным. Мы не можем обучить нашу нейронную сеть максимизировать ее с помощью градиентного спуска.

```
import editdistance


def get_distance(word, trans):
    """
    Функция, которая принимает слово и прогнозирует перевод
     и оценивает расстояние редактирования (Левенштейна) до ближайшего
правильного перевода
    """
    references = word_to_translation[word]
    assert len(references) != 0, "wrong/unknown word"
    return min(editdistance.eval(trans, ref) for ref in references)


def score(words, bsize=100):
    """функция, вычисляющая расстояние Левенштейна для случайных выборок
bsize"""
    assert isinstance(words, np.ndarray)

    batch_words = np.random.choice(words, size=bsize, replace=False)
    batch_trans = translate(batch_words)

    distances = list(map(get_distance, batch_words, batch_trans))

    return np.array(distances, dtype='float32')
```

```
# должно быть около 5-50 и быстро уменьшаться после тренировки
[score(test_words, 10).mean() for _ in range(5)]
```

Предварительное обучение под наблюдением

Здесь мы определяем функцию, которая обучает нашу модель за счет максимизации логарифмической вероятности, также известной как минимизация кроссэнтропии. Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```
# импорт служебных функций
from basic_model_tf import initialize_uninitialized, infer_length,
infer_mask, select_values_over_last_axis


class supervised_training:

    # переменная для ввода и правильных ответов
    input_sequence = tf.placeholder('int32', [None, None])
    reference_answers = tf.placeholder('int32', [None, None])

    # Вычислить логарифмические вероятности всех возможных токенов на каждом
шаге. Использовать интерфейс модели.
    logprobs_seq = <ВАШ КОД>
```

```
    # вычисление средней кроссэнтропии
    crossentropy = - select_values_over_last_axis(logprobs_seq,
reference_answers)

    mask = infer_mask(reference_answers, out_voc.eos_ix)

    loss = tf.reduce_sum(crossentropy * mask)/tf.reduce_sum(mask)

    # Оптимизатор веса сборки. Используйте model.weights, чтобы получить все
обучаемые параметры.
    train_step = <ВАШ КОД>


# инициализировать параметры оптимизатора, сохраняя при этом модель
нетронутой
initialize_uninitialized(s)
```

Актуальное обучение на минипакетах.

```
import random


def sample_batch(words, word_to_translation, batch_size):
    """
    выборка случайной партии слов и случайный правильный перевод для каждого
слова
     пример использования:
     batch_x,batch_y = sample_batch(train_words, word_to_translations,10)
    """
    # выбор слов
    batch_words = np.random.choice(words, size=batch_size)

    # выбор переводов
    batch_trans_candidates = list(map(word_to_translation.get, batch_words))
    batch_trans = list(map(random.choice, batch_trans_candidates))

    return inp_voc.to_matrix(batch_words), out_voc.to_matrix(batch_trans)
```

```
bx, by = sample_batch(train_words, word_to_translation, batch_size=3)
print("Source:")
print(bx)
print("Target:")
print(by)
```

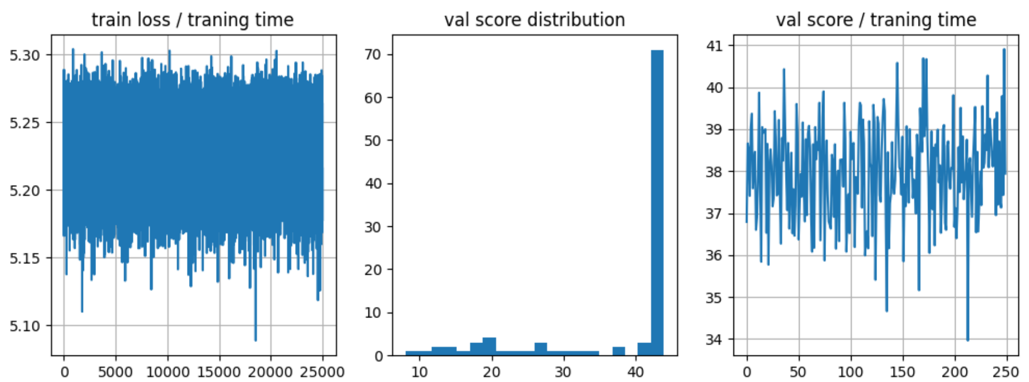
Вывод:

Source:
[[0 137 113 114 122 134 8 1 1 1 1 1 1 1 1]
 [0 116 127 122 121 137 122 2 138 127 122 113 136 117 1]
 [0 121 137 121 136 118 114 137 1 1 1 1 1 1 1]]
Target:
[[0 50 33 55 41 35 58 1 1 1 1 1 1 1 1 1 1]
 [0 36 45 41 52 50 57 2 51 40 37 45 57 33 43 33 1]
 [0 52 33 50 52 33 43 47 54 37 50 1 1 1 1 1 1]]

```
from tqdm import tqdm, trange  # либо tqdm_notebook,tnrange

loss_history = []
editdist_history = []

for i in trange(25000):
```

```
    bx, by = sample_batch(train_words, word_to_translation, 32)

    feed_dict = {
        supervised_training.input_sequence: bx,
        supervised_training.reference_answers: by
    }

    loss, _ = s.run([supervised_training.loss,
                     supervised_training.train_step], feed_dict)
    loss_history.append(loss)

    if (i+1) % REPORT_FREQ == 0:
        current_scores = score(test_words)
        editdist_history.append(current_scores.mean())
        plt.figure(figsize=(12, 4))
        plt.subplot(131)
        plt.title('train loss / traning time')
        plt.plot(loss_history)
        plt.grid()
        plt.subplot(132)
        plt.title('val score distribution')
        plt.hist(current_scores, bins=20)
        plt.subplot(133)
        plt.title('val score / traning time')
        plt.plot(editdist_history)
        plt.grid()
        plt.show()
        print("llh=%.3f, mean score=%.3f" %
              (np.mean(loss_history[-10:]), np.mean(editdist_history[-10:])))

# Примечание: это нормально, если потери колеблются вверх и вниз, если в
среднем есть улучшение в долгосрочной перспективе (например, партии по 5
тыс.)
```

Вывод:



100%|██████████| 25000/25000 [11:23<00:00, 36.56it/s]

llh=5.223, mean score=38.278

```
for word in train_words[:10]:
    print("%s -> %s" % (word, translate([word])[0]))
```

Вывод:

מגילת המקדש -> þþạạصęņπππŋŋņŋpņpķ?êêêảảrảr
תבנית:user ady -> ,тèèƛƛxņďťīpķ''' 老老ɳéééééé$$$$œ°
קטגוריה:שליטי איראן -> jźḥרצצםוfנשׁ7íssנגלôŝ星ôṣいmúåï'' 老老老ɳééééééé$
כתף-אום קרב -> þþþųصęņąąpгėŋŋņŋعęņ?שמצצנג
קנאבידיול -> @ṣкккккúû'' çéééŭöŭëmŭn

סקיילרק 1 -> dȟīâ"ńらら″チチב&&&‘ららķ榎榎
הדני המטבח:קטגוריה -> þþصąąpךėŋŋņŋ仙仙仙*••ø…?………ƅ…ƅ……t……ōōōaaů.
ד> ,- מורוזיני סקו'פרנצèèυëļţŋ仙仙ïḥ仙ïï a ::ævʊʊ$老老老°榎榎榎榎ɥěě۶۶۶
sysrq -> j$$œ$œ$œ御守守פפפ
רביע אבו-ח'ליל -> jűɔɔåɔộşåộş→ï→ïvvū````ąąąąpė-_

```
test_scores = []
for start_i in trange(0, len(test_words), 32):
    batch_words = test_words[start_i:start_i+32]
    batch_trans = translate(batch_words)
    distances = list(map(get_distance, batch_words, batch_trans))
    test_scores.extend(distances)

print("Supervised test score:", np.mean(test_scores))
```

Вывод:

Supervised test score: 41.44697202582232

4. Подготовка к обучению с подкреплением

Сначала нам нужно определить функцию потерь как пользовательскую операцию tf. Самый простой способ сделать это — использовать оболочку tensorflow.py_func.

```
def my_func(x):
  # x будет массивом numpy
  return np.sinh(x)
inp = tf.placeholder(tf.float32)
y = tf.py_func(my_func, [inp], tf.float32)
```

Ваша задача состоит в том, чтобы реализовать функцию _compute_levenshtein, которая принимает матрицы слов и переводов вместе с входными масками, затем преобразует их в фактические слова и фонемы и вычисляет min-levenshtein с помощью функции get_distance. Требуется дополнить код. Что должен выполнять код в каждой позиции <ВАШ КОД> указано в примечаниях к коду.

```

def _compute_levenshtein(words_ix, trans_ix):
    """
    Пользовательская операция тензорного потока, которая вычисляет потери
Левенштейна для прогнозируемого транса.

     Параметры:
     -words_ix - матрица буквенных индексов, shape=[batch_size,word_length]
     -words_mask - матрица нулей/единиц,
        1 означает "слово еще не закончено"
        0 означает, что "слово уже закончилось, и это заполнение"

     - trans_mask - матрица выходных буквенных индексов,
shape=[batch_size,translation_length]
     - trans_mask - матрица нулей/единиц, похожая на words_mask, но для
trans_ix


     Пожалуйста, реализуйте функцию и убедитесь, что она проходит тесты из
следующей ячейки.

    """

    # конвертировать слова в строки
```

```
    words = <ВАШ КОД: восстановить слова (список строк) из words_ix.
Использовать словарь>

    assert type(words) is list and type(
        words[0]) is str and len(words) == len(words_ix)

    # конвертировать переводы в список
    translations = <ВАШ КОД: восстановить trans (список списков фонем) из
trans_ix>

    assert type(translations) is list and type(
        translations[0]) is str and len(translations) == len(trans_ix)

    # вычислить расстояния Левенштейна. может быть произвольным кодом Python.
    distances = <ВАШ КОД: примените get_distance к каждой паре [words,
translations]>

    assert type(distances) in (list, tuple, np.ndarray) and len(
        distances) == len(words_ix)

    distances = np.array(list(distances), dtype='float32')
    return distances


def compute_levenshtein(words_ix, trans_ix):
    out = tf.py_func(_compute_levenshtein, [words_ix, trans_ix, ],
tf.float32)
    out.set_shape([None])

    return tf.stop_gradient(out)
```

Простой набор тестов, чтобы убедиться, что ваша реализация верна. Подсказка: если вы столкнетесь с какими-либо ошибками, не стесняйтесь использовать печать изнутри _compute_levenshtein.

```
# тесты
# выборка случайной партии (слова, правильный транс, неправильный транс)
batch_words = np.random.choice(train_words, size=100)
batch_trans = list(map(random.choice, map(
    word_to_translation.get, batch_words)))
batch_trans_wrong = np.random.choice(all_translations, size=100)

batch_words_ix = tf.constant(inp_voc.to_matrix(batch_words))
batch_trans_ix = tf.constant(out_voc.to_matrix(batch_trans))
batch_trans_wrong_ix = tf.constant(out_voc.to_matrix(batch_trans_wrong))

# calculate_levenshtein равен нулю для идеальных переводов
correct_answers_score = compute_levenshtein(
    batch_words_ix, batch_trans_ix).eval()

assert np.all(correct_answers_score ==
              0), "безупречный перевод получил ненулевой балл Левенштейна!"

print("Everything seems alright!")
```

Вывод:

Everything seems alright!

```
# compute_levenshtein соответствует фактической функции подсчета очков
wrong_answers_score = compute_levenshtein(
    batch_words_ix, batch_trans_wrong_ix).eval()
```

```
true_wrong_answers_score = np.array(
    list(map(get_distance, batch_words, batch_trans_wrong)))

assert np.all(wrong_answers_score ==
              true_wrong_answers_score), "для некоторого слова символическое
расстояние Левенштейна отличается от фактического расстояния Левенштейна"

print("Everything seems alright!")
```

Вывод:

Everything seems alright!

5. Градиент самокритичной политики

В этом разделе вы реализуете алгоритм, называемый обучением самокритичной последовательности. Алгоритм представляет собой ванильный градиент политики со специальной базовой линией. Алгоритм представляет собой ванильный градиент политики со специальной базовой линией.

$$\nabla J = E_{x\sim p(s)} E_{y\sim \pi(y|x)} \nabla log \pi(y|x) \cdot (R(x,y) - b(x))$$

Здесь вознаграждение R(x,y) является отрицательным расстоянием Левенштейна (поскольку мы его минимизируем). Базовый показатель b(x) показывает, насколько хорошо модель справляется со словом x. На практике это означает, что мы вычисляем базовый уровень как показатель жадного перевода,

$$b(x) = R(x, y_{greedy}(x)).$$

Отметим, что мы уже получили необходимые выходные данные: model.greedy_translations, model.greedy_mask, и нам нужно только вычислить левенштейн с помощью функции calculate_levenshtein.

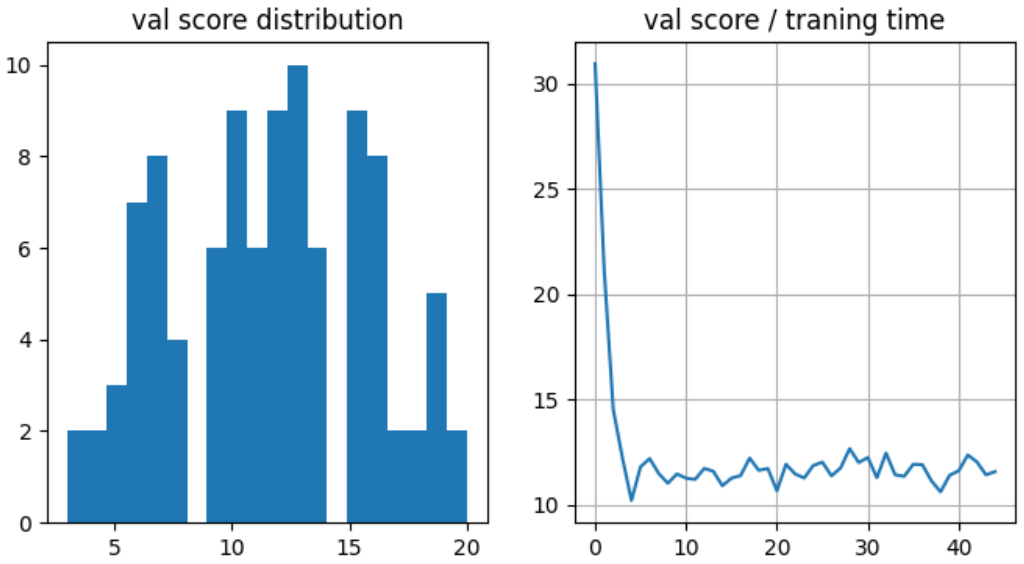
```
class trainer:

    input_sequence = tf.placeholder('int32', [None, None])

    # использовать модель для __sample__ символических переводов с учетом
input_sequence
    sample_translations, sample_logp = <ВАШ КОД>
    # использовать модель для __greedy__ символических переводов, заданных
input_sequence
    greedy_translations, greedy_logp = <ВАШ КОД>

    rewards = - compute_levenshtein(input_sequence, sample_translations)

    # вычислить __negative__ levenshtein для жадного режима
    baseline = <ВАШ КОД>

    # вычислить преимущество, используя вознаграждение и базовый уровень
    advantage = <ВАШ КОД: вычислить аознаграждение>
    assert advantage.shape.ndims == 1, "преимущество должно быть в форме
[batch_size]"

    # вычислить log_pi(a_t|s_t), shape = [batch, seq_length]
    logprobs_phoneme = <ВАШ КОД>
    # ^-- подсказка: посмотрите, как реализована кроссэнтропия в
```

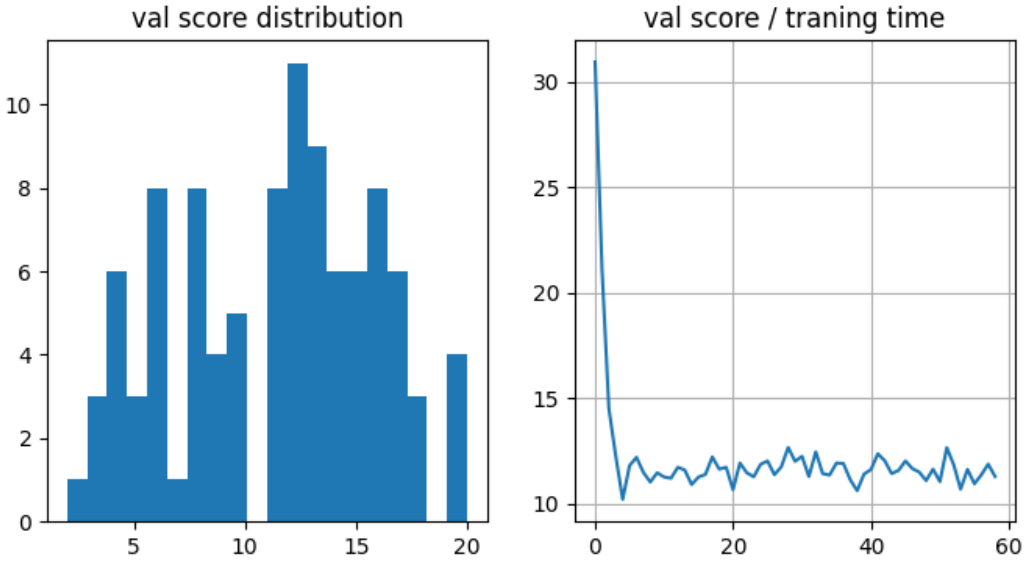
```
контролируемой потере обучения выше
    # обратите внимание на знак - его нельзя умножать на -1 :)

    # Градиент политики вычислений
    # или, скорее, суррогатная функция, чей градиент является градиентом
политики
    J = logprobs_phoneme*advantage[:, None]

    mask = infer_mask(sample_translations, out_voc.eos_ix)
    loss = - tf.reduce_sum(J*mask) / tf.reduce_sum(mask)

    # регуляризация с отрицательной энтропией. Не забудьте знак!
    # примечание: для энтропии вам нужны вероятности для всех токенов
(sample_logp), а не только для phoneme_logprobs
    entropy = <ВЫШ КОД: вычислить энтропийную матрицу формы[пакет, длина
последовательности], H = -sum(p*log_p), не забудьте знак!>
    # подсказка: вы можете получить выборочные вероятности из sample_logp,
используя математику :)


    assert entropy.shape.ndims == 2, "пожалуйста, убедитесь, что поэлементная
энтропия имеет форму [batch,time]"

    loss -= 0.01*tf.reduce_sum(entropy*mask) / tf.reduce_sum(mask)

    #вычислить обновления веса, обрезать по норме
    grads = tf.gradients(loss, model.weights)
    grads = tf.clip_by_global_norm(grads, 50)[0]

    train_step = tf.train.AdamOptimizer(
        learning_rate=1e-5).apply_gradients(zip(grads, model.weights,))


initialize_uninitialized()
```

Обучение градиенту политики

```
loss_history = []
editdist_history = []

for i in trange(10000):
    bx = sample_batch(train_words, word_to_translation, 32)[0]
    pseudo_loss, _ = s.run([trainer.loss, trainer.train_step], {
                            trainer.input_sequence: bx})

    loss_history.append(
        pseudo_loss
    )

    if (i+1) % REPORT_FREQ == 0:
        current_scores = score(test_words)
        editdist_history.append(current_scores.mean())
        plt.figure(figsize=(8, 4))
        plt.subplot(121)
        plt.title('val score distribution')
        plt.hist(current_scores, bins=20)
        plt.subplot(122)
        plt.title('val score / traning time')
        plt.plot(editdist_history)
        plt.grid()
        plt.show()
        print("J=%.3f, mean score=%.3f" %
              (np.mean(loss_history[-10:]), np.mean(editdist_history[-10:])))
```

Вывод:





100%|██████████| 10000/10000 [17:46<00:00, 9.38it/s]

J=-0.103, mean score=10.657

```
for word in train_words[:10]:
    print("%s -> %s" % (word, translate([word])[0]))

test_scores = []
for start_i in trange(0, len(test_words), 32):
    batch_words = test_words[start_i:start_i+32]
    batch_trans = translate(batch_words)
    distances = list(map(get_distance, batch_words, batch_trans))
    test_scores.extend(distances)
print("Supervised test score:", np.mean(test_scores))
```

Вывод:

המקדש מגילת -> aie aie

תבנית:user ady -> aie aie
איראן שליטי:קטגוריה -> aie aiea
בתף-אום קרב -> aie aie
קנאבידיול -> aie aie
סקיילרק 1 -> aie aie
הדני המטבח:קטגוריה -> aie aie
מורוזיני סקו'פרנצ -> aie a aie
sysrq -> aie
ליל'ח-אבו רביע -> aie a aie

Задания к лабораторной работе №1

1. Написать свой код к разделам 1 - 5 согласно заданиям для всех фрагментов, помеченных как: <ВАШ КОД:>

```
<ВАШ КОД: >
```

2. Запустить полученную программу, получить ожидаемые выходные данные.

Задания к лабораторной работе №2

1. Замените библиотеку Tensorflow на Pytorch и воспроизведите все эксперименты из разделов 1-5.

Требование к отчету

1. В качестве отчета принимаются два Python файла с кодом (для Tensorflow и Pytorch).
2. Код должен быть в достаточной мере прокомментирован.
3. Отчет предоставляется в виде архива zip, формат имени архива: <номер группы>_<Фамилия_Имя>_LR<номер работы>.zip