Вопрос 1

Верно

Баллов: 1,00 из 1,00

Напишите функцию, реализующую max-pooling с заданной шириной окна. В качестве пособия можно использовать другую задачу.

Функция должна возвращать два тензора одинаковой размерности $OutLen \times EmbSize$:

- первый тензор - основной результат - результат применения max-пулинга к каждому столбцу для каждой позиции скользящего окна
- второй тензор - информация, нужная для расчёта градиента, - относительные индексы максимальных элементов для каждого столбца для каждой позиции скользящего окна

Например, для фрагмента входной матрицы $\begin{pmatrix} 1 & 0 & 3 \\ 0 & 1 & 4 \end{pmatrix}$ и размера скользящего окна $k = 2$ первый результат должен иметь вид $result = \begin{pmatrix} 1 & 1 & 4 \end{pmatrix}$, а второй $indices = \begin{pmatrix} 0 & 1 & 1 \end{pmatrix}$. Индексы - номера строк относительно позиции окна ($0 \le indices_i < k$), из которых были взяты соответствующие элементы $result$

Ответ: (штрафной режим: 0 %)

Сброс ответа

```python
import sys
import ast
import numpy as np


def parse_array(s):
    return np.array(ast.literal_eval(s))

def read_array():
    return parse_array(sys.stdin.readline())

def write_array(arr):
    print(repr(arr.tolist()))


def max_pooling(features, kernel_size):
    """
    features - InLen x EmbSize - features of elements of input sequence
    kernel_size - positive integer - size of sliding window

    returns tuple of two matrices of shape OutLen x EmbSize:
         - output features (main result)
         - relative indices of maximum elements for each position of sliding window
    """
    assert kernel_size > 0 and isinstance(kernel_size, int), "kernel_size must be a positive integer"

    # Размеры входных данных
    in_len, emb_size = features.shape

    # Вычисляем размер выходных данных
    out_len = in_len - kernel_size + 1

    # Инициализируем матрицы для выходных данных и индексов максимальных элементов
    output_features = np.zeros((out_len, emb_size))
    max_indices = np.zeros((out_len, emb_size), dtype=int)

    # Проходим по каждому окну
    for i in range(out_len):
        window = features[i:i+kernel_size, :]
        max_values = np.max(window, axis=0)
        output_features[i, :] = max_values

        # Находим индексы максимальных элементов в окне
        max_indices[i, :] = np.argmax(window, axis=0)

    return output_features, max_indices


features = read_array()
kernel_size = int(sys.stdin.readline())

result, indices = max_pooling(features, kernel_size)
```

Вопрос 2

Верно

Баллов: 1,00 из 1,00

Прямому проходу по модулю max-пулинга была посвящена предыдущая задача.

Теперь займёмся обратным проходом: напишите функцию, вычисляющую **производную функции потерь по входам модуля max-пулинга** $\frac{\partial Loss}{\partial features}$.

Функция принимает следующие аргументы:

- $features \in \mathbb{R}^{InLen \times EmbSize}$ - признаки, которые были переданы на вход модулю при прямом проходе
- $2 \leq kernel\_size \leq InLen$ - размер скользящего окна
- $indices \in \mathbb{N}^{OutLen \times EmbSize}, \quad 0 \leq indices < kernel\_size$ - относительная позиция максимального элемента внутри скользящего окна для каждого элемента выходного тензора (смещение относительно номера строки)
- $dldout = \frac{\partial Loss}{\partial out} \in \mathbb{R}^{OutLen \times EmbSize}$ - значение производной функции потерь по выходам слоя max-пулинга (то есть производная по входам следующего слоя)

Вам может быть полезна формула производной кусочно-линейной функции $ReLU(x) = \begin{cases} 0 & if \quad x \leq 0 \\ x & otherwise \end{cases}: \frac{\partial ReLU(x)}{\partial x} = \begin{cases} 0 & if \quad x \leq 0 \\ 1 & otherwise \end{cases}$

А также помните про правило цепочки: $\frac{\partial Loss}{\partial features} = \frac{\partial Loss}{\partial out} \frac{\partial out}{\partial features}$.

Ответ: (штрафной режим: 0 %)

Сброс ответа

```python
import sys
import ast
import numpy as np


def parse_array(s):
    return np.array(ast.literal_eval(s))

def read_array():
    return parse_array(sys.stdin.readline())

def write_array(arr):
    print(repr(arr.tolist()))


def max_pooling_dldfeatures(features, kernel_size, indices, dldout):
    """
    features - InLen x EmbSize - features of elements of input sequence
    kernel_size - positive integer - size of sliding window
    indices - OutLen x EmbSize - relative indices of maximum elements for each window position
    dldout - OutLen x EmbSize - partial derivative of loss function with respect to outputs of max_pooling layer

    returns InLen x EmbSize
    """
    dldfeatures = np.zeros_like(features)

    # Размеры входных данных
    in_len, emb_size = features.shape

    # Вычисление количества окон
    num_windows = in_len - kernel_size + 1

    # Проходим по каждому окну
    for i in range(num_windows):
        # Находим индексы максимальных элементов в текущем окне
        max_indices = indices[i]

        # Применяем градиенты к соответствующим признакам
        for j in range(emb_size):
            # Создаем маску для текущего окна
            mask = np.zeros(kernel_size)
            mask[max_indices[j]] = 1
            # Применяем градиент к максимальному элементу в окне
            dldfeatures[i:i+kernel_size, j] += dldout[i, j] * mask

    return dldfeatures


features = read_array()
kernel_size = int(sys.stdin.readline())
indices = read_array().astype('uint32')
dldout = read_array()
```

Вопрос 3

Верно

Баллов: 1,00 из 1,00

Вектор-функция $softmax(x) = \left(\frac{e^{x_1}}{\sum_j e^{x_j}} \quad \cdots \quad \frac{e^{x_n}}{\sum_j e^{x_j}} \right)$ - популярный способ нормировать вектор чисел $x \in \mathbb{R}^n$ так, чтобы $0 \leq softmax_i(x) \leq 1$ и $\sum_i softmax_i(x) = 1$.

Напишите функцию, вычисляющую softmax для заданного вектора.

Используйте экспоненту из numpy: np.exp(x).

Ответ: (штрафной режим: 0 %)

Сброс ответа

```
import sys
import ast
import numpy as np


def parse_array(s):
    return np.array(ast.literal_eval(s))

def read_array():
    return parse_array(sys.stdin.readline())

def write_array(arr):
    print(repr(arr.tolist()))


def softmax(x):
    """
    x - vector of n elements - input

    returns vector of n elements - softmax output
    """
    exp_values = np.exp(x)
    # Вычисляем сумму всех экспоненциальных значений
    exp_values_sum = np.sum(exp_values)
    # Возвращаем результат деления каждого экспоненциального значения на сумму всех экспоненциальных значений
    return exp_values / exp_values_sum


x = read_array()

result = softmax(x)

write_array(result)

```

	Input	Got	Expected	Comment
✔	[0.8256360412886801, -1.0640429421088764, 0.06570993778672017, -0.2494659991276688, 1.0091866702514656, -0.8673301281740561, 2.473160080998488, 1.31267178262454, 0.2477748733331122]	[0.09321212094746084, 0.014086244909761809, 0.04359540098595044, 0.03180984159112158, 0.11199210696472814, 0.017148515272924707, 0.4841534035729015, 0.1517013078570136, 0.052301057898137465]		Right answer!
✔	[-0.5573762774997593, -0.12486952307645574, -0.4998257879074948, -0.4862776568987329, 0.3163493684507575, 1.2646796053617368, 0.7855635616198732]	[0.058531870315824304, 0.09020439509944926, 0.06199922513896919, 0.06284491458367257, 0.14023183868371222, 0.36199387090387236, 0.22419388527450007]		Right answer!
✔	[0.32986479588396106, -1.3868999798143808, 1.2691468860308528, 1.3528212486900757, -1.3931448512349935, -0.3352799714607657, 0.1298917945352221, 1.0734877106856757, 0.9346917480976176, -0.11098614228708285, -0.4648043295430369, -0.5898088541354348, -0.13502630588301265, 0.8878115931253887, -0.8440547707394371, 0.43068397113510254]	[0.057972569084937406, 0.010414563635733952, 0.1483021937978658, 0.16124523497014187, 0.010349728678617314, 0.029809441867398246, 0.04746520662403762, 0.12194777345724596, 0.10614402561354713, 0.03730468842120642, 0.026187987844961354, 0.023110713591305846, 0.03641857148243176, 0.10128281468180826, 0.017922383118655575, 0.06412210313010554]		Right answer!
✔	[1.111667959020268, -0.8113065864139262, -1.0418003238358182, 0.8999342066667645, -0.5902574539585203, -0.7399078145377962, 0.29749176485750195]	[0.3504150964522404, 0.05122070763024041, 0.040676484894636884, 0.28354892670943627, 0.06389192784744997, 0.05501152276230134, 0.1552353337036948]		Right answer!

Вопрос 4

Верно

Баллов: 1,00 из 1,00

Напишите функцию, реализующую производную выхода softmax по входам $\frac{\partial softmax(x)}{\partial x}$.

Помните, что и $softmax$ и x - вектора из n элементов, поэтому производная - это матрица, в ij-ячейке которой стоит производная i-го элемента $softmax_i(x)$ по j-му входу x_j (i соответствует номеру строки, j - номер столбца). Такая матрица ещё называется матрицей Якоби.

Подсказка: возможно, будет проще решать эту задачу, рассматривая два случая, когда $i = j$ и когда $i \neq j$.

Ответ: (штрафной режим: 0 %)

Сброс ответа

```
import sys
import ast
import numpy as np


def parse_array(s):
    return np.array(ast.literal_eval(s))

def read_array():
    return parse_array(sys.stdin.readline())

def write_array(arr):
    print(repr(arr.tolist()))


def dsoftmax_dx(x):
    """
    x - vector of n elements - input

    returns matrix n x n
    """
    exp_values = np.exp(x)
    exp_values_sum = np.sum(exp_values)
    softmax_x = exp_values / exp_values_sum

    # Инициализируем матрицу для хранения градиентов размером n x n
    dsoftmax_dx = np.zeros((len(x), len(x)))

    # Вычисляем градиенты для каждого элемента входного вектора
    for i in range(len(x)):
        for j in range(len(x)):
            if i == j:
                # Для диагональных элементов матрицы градиент равен softmax_x[i] * (1 - softmax_x[i])
                dsoftmax_dx[i, j] = softmax_x[i] * (1 - softmax_x[i])
            else:
                # Для остальных элементов матрицы градиент равен -softmax_x[i] * softmax_x[j]
                dsoftmax_dx[i, j] = -softmax_x[i] * softmax_x[j]

    return dsoftmax_dx


x = read_array()

result = dsoftmax_dx(x)

write_array(result)

```

Вопрос 5

Верно

Баллов: 1,00 из 1,00

Напишите функцию, реализующую механизм внимания, как в этой задаче.

Ответ: (штрафной режим: 0 %)

Сброс ответа

```
import sys
import ast
import numpy as np


def parse_array(s):
    return np.array(ast.literal_eval(s))

def read_array():
    return parse_array(sys.stdin.readline())

def write_array(arr):
    print(repr(arr.tolist()))


def attention(features, query):
    """
    features - InLen x EmbSize - features of elements of input sequence
    query - EmbSize - features of query object

    returns vector of size EmbSize - features, aggregated according to the query
    """
    unScores = features@query
    exp_values = np.exp(unScores)
    exp_values_sum = np.sum(exp_values)
    attScores = exp_values / exp_values_sum
    result = np.multiply(features, attScores[:, None])
    column_sums = np.sum(result, axis=0)
    return column_sums


features = read_array()
query = read_array()

result = attention(features, query)

write_array(result)

```

Вопрос 6

Неверно

Баллов: 0,00 из 1,00

Напишите функцию, реализующую механизм self-attention внимания с линейными преобразованиями, как в этой задаче.

Ответ: (штрафной режим: 0 %)

Сброс ответа

```
import sys
import ast
import numpy as np


def parse_array(s):
    return np.array(ast.literal_eval(s))

def read_array():
    return parse_array(sys.stdin.readline())

def write_array(arr):
    print(repr(arr.tolist()))


def self_attention(features, proj_k, bias_k, proj_q, bias_q, proj_v, bias_v):
    """
    features - InLen x EmbSize - features of elements of input sequence
    proj_k - EmbSize x EmbSize - projection matrix to make keys from features
    bias_k - EmbSize - bias vector to make keys from features
    proj_q - EmbSize x EmbSize - projection matrix to make queries from features
    bias_q - EmbSize - bias vector to make queries from features
    proj_v - EmbSize x EmbSize - projection matrix to make values from features
    bias_v - EmbSize - bias vector to make values from features

    returns InLen x EmbSize
    """
    Keys = features@proj_q+bias_q
    Queries = features@proj_k+bias_k
    Values = features@proj_v+bias_v
    Logists = Queries@Keys.T
    AttScores = softmax(Logists)
    s = AttScores @ Values
    return s

def softmax(x):
    max_x = np.amax(x, axis=1).reshape(x.shape[0], 1) # Находим максимальное значение в каждой строке
    e_x = np.exp(x - max_x) # Вычитаем максимальное значение для стабильности
    return e_x / e_x.sum(axis=1, keepdims=True) # Делим на сумму по строкам

features = read_array()
proj_k = read_array()
bias_k = read_array()
proj_q = read_array()
bias_q = read_array()
proj_v = read_array()
bias_v = read_array()

result = self_attention(features, proj_k, bias_k, proj_q, bias_q, proj_v, bias_v)

write_array(result)

```