Практическая работа №4

Выполнил студент группы БВТ2003 Глазков Даниил

Задача 1

Необходимо реализовать нейронную сеть вычисляющую результат заданной логической операции. Затем реализовать функции, которые будут симулировать работу построенной модели. Функции должны принимать тензор входных данных и список весов. Должно быть реализовано 2 функции:

Функция, в которой все операции реализованы как поэлементные операции над тензорами

Функция, в которой все операции реализованы с использованием операций над тензорами из NumPy

Для проверки корректности работы функций необходимо:

Инициализировать модель и получить из нее веса

Прогнать датасет через не обученную модель и реализованные 2 функции. Сравнить результат.

Обучить модель и получить веса после обучения

Прогнать датасет через обученную модель и реализованные 2 функции. Сравнить результат.

In [1]:
```
import os

import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
import matplotlib.pyplot as plt
from matplotlib import gridspec

activation = {'sigmoid': lambda x: 1 / (1 + np.exp(-x)),
              'relu': lambda x: np.maximum(x, 0)}

def plots(H):
    loss = H.history['loss']
    acc = H.history['accuracy']
    epochs = range(1, len(loss) + 1)
    fig = plt.figure(figsize=(12,6))
    gs = gridspec.GridSpec(1, 2, width_ratios=[3, 3])
    plt.subplot(gs[0])
    plt.plot(epochs, loss, 'r*-', label='Training loss')
    plt.title('Training loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
```

```
    plt.subplot(gs[1])
    plt.plot(epochs, acc, 'b*-', label='Training acc')
    plt.title('Training accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.tight_layout()
    plt.show()

def naive_dot_vXv(x, y):
    assert len(x.shape) == 1
    assert len(y.shape) == 1
    assert x.shape[0] == y.shape[0]
    z = 0.
    for i in range(x.shape[0]):
        z += x[i] * y[i]
    return z

def naive_dot_vXm(y, x):
    z = np.zeros(x.shape[1])
    for i in range(x.shape[1]):
        z[i] = naive_dot_vXv(y, x[:, i])
    return z

def naive_add_vXv(x, y):
    assert len(x.shape) == 1
    assert x.shape == y.shape
    x = x.copy()
    for i in range(x.shape[0]):
        x[i] += y[i]
    return x

def sol(x, y, z):
    f = x and y
    ff = x and z
    return f or ff

def numpy_sol(W, B, input):
    x = input.copy()
    for i in range(len(W)):
        x = np.dot(x, W[i])
        x += B[i]
        x = activation['relu'](x) if i != range(len(W))[-1] else activation['sig
    return x

def native_sol(W, B, input):
    x = input.copy()
    for i in range(len(W)):
        x = np.array([naive_dot_vXm(el, W[i]) for el in x])
        x = np.array([naive_add_vXv(el, B[i]) for el in x])
        x = [activation['relu'](el) for el in x] if i != range(len(W))[-1] else
    return np.array(x)

def solution(input, model):
    W = [layer.get_weights()[0] for layer in model.layers]
    B = [layer.get_weights()[1] for layer in model.layers]
    # layer_names = [layer.name for layer in model.layers]
    print(f'predict:\n{model.predict(input)}')
    print(f'numpy:\n{numpy_sol(W, B, input)}')
    print(f'native:\n{native_sol(W, B, input)}')
```

```
x = np.array([[0, 0, 0], [0, 0, 1], [0, 1, 0], [0, 1, 1],
              [1, 0, 0], [1, 0, 1], [1, 1, 0], [1, 1, 1]])
y = np.array([sol(i[0],i[1],i[2]) for i in x])

model = Sequential()
model.add(Dense(16, activation='relu', input_dim=3))
model.add(Dense(8, activation='relu'))
model.add(Dense(8, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['accurac
print('Необученная модель\n----------------------')
solution(x, model)
H = model.fit(x, y, epochs=100, batch_size=1, verbose=0)
print('\nОбученная модель\n----------------------')
solution(x, model)
plots(H)
```

```
Необученная модель
----------------------
1/1 [==============================] - 0s 140ms/step
predict:
[[0.5       ]
 [0.51084906]
 [0.515769  ]
 [0.52288413]
 [0.5154182 ]
 [0.51802963]
 [0.52880245]
 [0.5325948 ]]
numpy:
[[0.5       ]
 [0.51084906]
 [0.51576899]
 [0.52288414]
 [0.51541815]
 [0.51802967]
 [0.52880247]
 [0.5325948 ]]
native:
[[0.5       ]
 [0.51084906]
 [0.51576899]
 [0.52288414]
 [0.51541815]
 [0.51802967]
 [0.52880247]
 [0.5325948 ]]

Обученная модель
----------------------
1/1 [==============================] - 0s 24ms/step
predict:
[[0.08551234]
 [0.03525427]
 [0.05565871]
 [0.04220378]
 [0.5368474 ]
 [0.76938087]
 [0.8070266 ]
 [0.84964246]]
numpy:
[[0.08551232]
 [0.03525426]
 [0.05565871]
 [0.04220376]
 [0.53684746]
 [0.76938092]
 [0.80702663]
 [0.84964244]]
native:
[[0.08551232]
 [0.03525426]
 [0.05565871]
 [0.04220376]
 [0.53684746]
 [0.76938092]
```

```
[0.80702663]
[0.84964244]]
```