Реализация 1 для NNI по статье: Using Neural Networks for Fast Numerical Integration and Optimization

Цели

1. без углубления в детали повторить идею, описанную в статье,
2. проанализировать результаты.

Материалы

Статья: Using Neural Networks for Fast Numerical Integration and Optimization

Использование полилогарифмической функции библиотеки mpmath

Импорт библиотек

In [8]:
```
import numpy as np
import matplotlib.pyplot as plt
```

In [9]:
```
from keras.models import Sequential
from keras.layers import Dense
from sklearn.model_selection import train_test_split
from scipy.special import expi
```

In [10]:
```
from scipy.integrate import simpson
```

In [11]:
```
from mpmath import polylog
```

In [12]:
```
from math import sin, cos
```

Подинтегральная функция

In [13]:
```
def f(x):
  return np.log(x)
```

Функция генерации общей выборки

In [14]:
```
def generate_dataset(func, size):
    x = np.random.uniform(0, 100, size)
```

```
    y = f(x)
    data_set = np.column_stack((x, y))

    return data_set
```

Численный метод Симпсона (на основе библиотеки scipy)

In [15]:
```
def solve_integral(a, b):
    x = np.linspace(a, b, 100)
    y = f(x)
    dx = (b - a) / 99
    integral = simpson(y, dx=dx)
    return integral
```

Реализация интегрированной нейросети для NNI

In [16]:
```
def sigmoid_integral_term(b_1_j, w_1_j, alpha_1, beta_1):
    term1 = polylog(1, -np.exp(-b_1_j - w_1_j * alpha_1))
    term2 = polylog(1, -np.exp(-b_1_j - w_1_j * beta_1))

    return term1 - term2
```

In [17]:
```
def approximated_integral(weights_and_biases, alpha_1, beta_1):

    w_1 = list(weights_and_biases[0][0])
    b_1 = list(weights_and_biases[1])
    w_2 = list(weights_and_biases[2])
    b_2 = list(weights_and_biases[3])

    integral_value = b_2 * (beta_1 - alpha_1)

    for j in range(len(w_2)):
        Phi_j = sigmoid_integral_term(b_1[j], w_1[j], alpha_1, beta_1)
        integral_value += w_2[j] * ((beta_1 - alpha_1) + Phi_j / w_1[j])

    return integral_value
```

Создание выборки, разделение на обучающую и тестовую выборки

Совокупность состоит и 100 000 пар чисел (x, f(x)), где x - случайное число (вероятность распределена равномерно), f(x) - подинтегральная функция от переменной x, определенная выше.

In [18]:
```
dataset = generate_dataset(f, 1000000)
dataset.shape
```

Out[18]: (1000000, 2)

In [19]:
```
dataset[46]
```

Out[19]: array([69.584195 , 4.24253746])

In [20]:
```
x = dataset[:, 0]
y = dataset[:, 1]
```

In [21]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25)
```

In [22]:
```
print(x_train.shape, y_train.shape, x_test.shape, y_test.shape)
```

```
(750000,) (750000,) (250000,) (250000,)
```

Создание, компиляция и обучение многослойного перцептрона по архитектуре описанной в статье

!!! В отличие от статьи, был использован имеющийся в библиотеке keras оптимизатор Adam, а не предложенная в статье Levenberg-Marquardt, что будет изменено в следующих итерациях работы над проектом.

На втором слое используется функция активации - логистическая сигмоида (в keras она носит название sigmoid), интеграл которой в контексте полилогарифмической функции описан в статье формулами (9-11).

На третьем слое используется функция активации linear, которая возвращает значение без измнений.

In [23]:
```
model = Sequential()
model.add(Dense(10000, input_dim=1, activation='sigmoid'))
model.add(Dense(1, activation='linear'))
model.summary()
```

```
/Users/grindelf/Programming/JUPITER/venv/lib/python3.12/site-packages/keras/src/layers/core/dense.py:87: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

Model: "sequential"

Layer (type)	Output Shape	Par
dense (Dense)	(None, 10000)	20
dense_1 (Dense)	(None, 1)	10

Total params: 30,001 (117.19 KB)

Trainable params: 30,001 (117.19 KB)

Non-trainable params: 0 (0.00 B)

In [24]:
```
model.compile(optimizer='adam',
              loss='mean_squared_error')
```

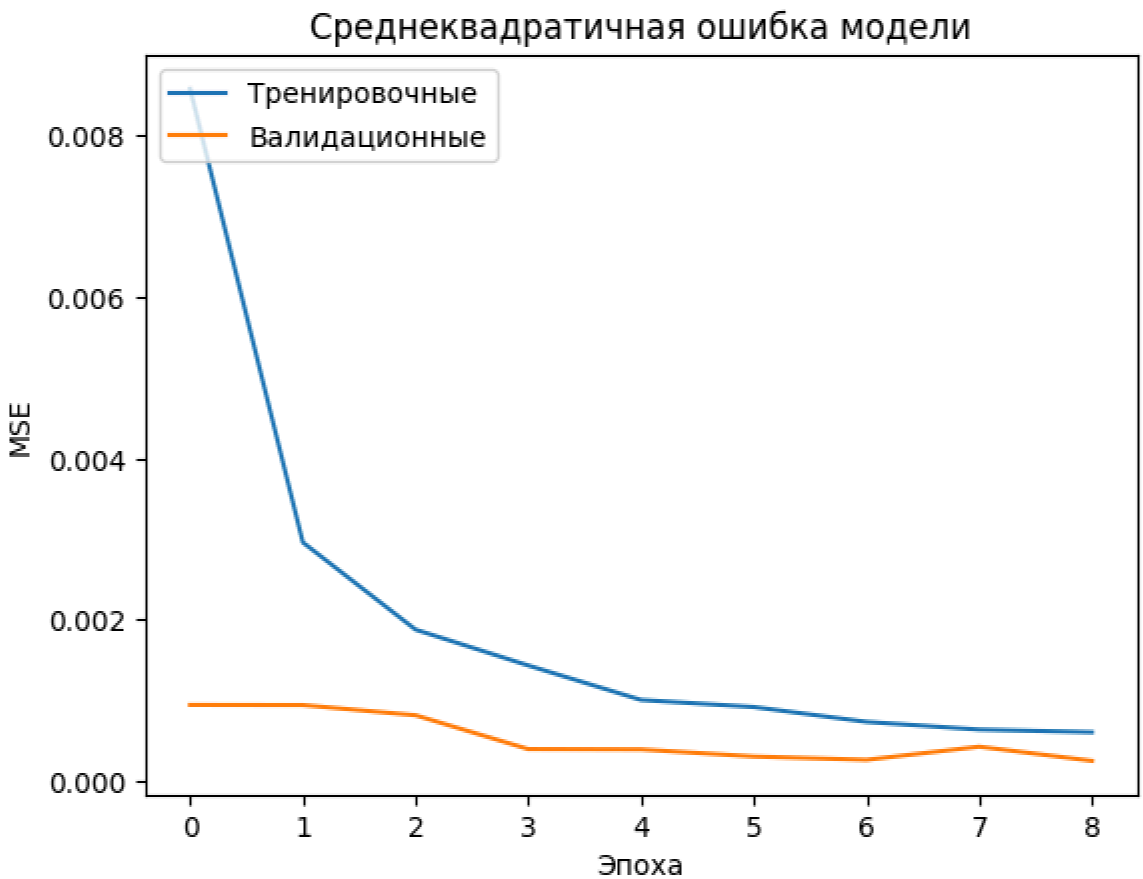
Обучение проводится с размером батча 20, в 30 эпох. 33% тренировочной выборки используется для валидации.

In [25]:
```
history = model.fit(x_train, y_train, batch_size=20, epochs=10, validation_s
```

```
Epoch 1/10
25125/25125 ━━━━━━━━━━━━━━━━━━━━ 31s 1ms/step - loss: 0.0738 - val_loss: 0.0
120
Epoch 2/10
25125/25125 ━━━━━━━━━━━━━━━━━━━━ 29s 1ms/step - loss: 0.0137 - val_loss: 9.4
388e-04
Epoch 3/10
25125/25125 ━━━━━━━━━━━━━━━━━━━━ 29s 1ms/step - loss: 0.0035 - val_loss: 9.4
162e-04
Epoch 4/10
25125/25125 ━━━━━━━━━━━━━━━━━━━━ 31s 1ms/step - loss: 0.0019 - val_loss: 8.1
536e-04
Epoch 5/10
25125/25125 ━━━━━━━━━━━━━━━━━━━━ 33s 1ms/step - loss: 0.0016 - val_loss: 3.9
602e-04
Epoch 6/10
25125/25125 ━━━━━━━━━━━━━━━━━━━━ 32s 1ms/step - loss: 9.5622e-04 - val_loss:
3.9207e-04
Epoch 7/10
25125/25125 ━━━━━━━━━━━━━━━━━━━━ 32s 1ms/step - loss: 0.0011 - val_loss: 3.0
457e-04
Epoch 8/10
25125/25125 ━━━━━━━━━━━━━━━━━━━━ 31s 1ms/step - loss: 8.3983e-04 - val_loss:
2.6296e-04
Epoch 9/10
25125/25125 ━━━━━━━━━━━━━━━━━━━━ 32s 1ms/step - loss: 6.1866e-04 - val_loss:
4.2634e-04
Epoch 10/10
25125/25125 ━━━━━━━━━━━━━━━━━━━━ 32s 1ms/step - loss: 6.4321e-04 - val_loss:
2.5071e-04
```

Оценка обучения нейросети по фунции потерь "Среднеквадратичная ошибка"

In [26]:
```
plt.plot(history.history['loss'][1:])
plt.plot(history.history['val_loss'][1:])
plt.title('Среднеквадратичная ошибка модели')
plt.ylabel('MSE')
plt.xlabel('Эпоха')
plt.legend(['Тренировочные', 'Валидационные'], loc='upper left')
plt.show()
```



Тестирование модели

Результат ниже в виде функции потерь

In [27]:
```
model.evaluate(x_test, y_test)
```

```
7813/7813 ━━━━━━━━━━━━━━━━━━━━ 5s 593us/step - loss: 4.4690e-04
```

Out[27]: `0.00048760388744995`

Использование весов и смещений обученной нейросети для вычисления определенных интегралов по различным интервалам

In [37]:
```
nni = []

w_b = model.get_weights()

for i in range(0, 50, 5):
  nni.append(approximated_integral(w_b, i, 3*i + 5)[0])
  print(f'Интеграл между {i} и {3*i + 5}: {nni[-1]}')
```

```
Интеграл между 0 и 5: 3.93645814556901
Интеграл между 5 и 20: 39.7984886815476
Интеграл между 10 и 35: 81.4481039637994
Интеграл между 15 и 50: 127.085895555461
/var/folders/df/v02b0vfx2tbc5plrdx4v6gk80000gn/T/ipykernel_14537/3468556677.
py:3: RuntimeWarning: overflow encountered in exp
  term2 = polylog(1, -np.exp(-b_1_j - w_1_j * beta_1))
Интеграл между 20 и 65: +inf
Интеграл между 25 и 80: +inf
Интеграл между 30 и 95: +inf
Интеграл между 35 и 110: nan
Интеграл между 40 и 125: nan
Интеграл между 45 и 140: nan
```

Вычисление определенных интегралов по подобным интервалам методом Симпсона

In [38]:
```
sni = []

for i in range(0, 50, 5):
  sni.append(solve_integral(i, 3*i + 5))
  print(f'Интеграл между {i} и {3*i + 5}: {sni[-1]}')
```

```
Интеграл между 0 и 5: -inf
Интеграл между 5 и 20: 36.86745586847303
Интеграл между 10 и 35: 76.41133118617802
Интеграл между 15 и 50: 119.98039721551218
Интеграл между 20 и 65: 166.42052702784707
Интеграл между 25 и 80: 215.0902351024121
Интеграл между 30 и 95: 265.58238319661587
Интеграл между 35 и 110: 317.615658023537
Интеграл между 40 и 125: 370.9840389307101
Интеграл между 45 и 140: 425.53012705203713
/var/folders/df/v02b0vfx2tbc5plrdx4v6gk80000gn/T/ipykernel_14537/3039598682.
py:2: RuntimeWarning: divide by zero encountered in log
  return np.log(x)
```

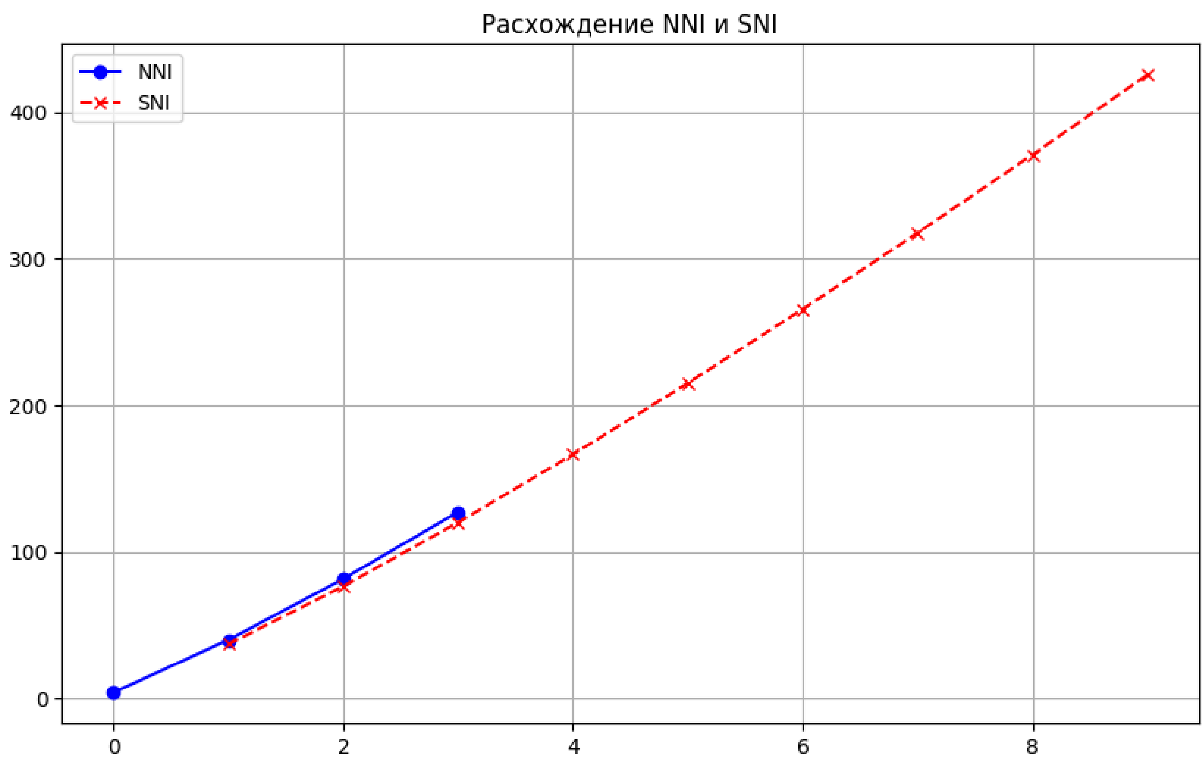
Абсолютная величина расхождения значений определенных интеграллов полученных двумя методами и визуализация

In [39]:
```
for i in range(len(nni)):
  print(abs(nni[i] - sni[i]))
```

```
+inf
2.93103281307453
5.03677277762137
7.10549833994887
+inf
+inf
+inf
nan
nan
nan
```

In [40]:
```python
plt.figure(figsize=(10, 6))
plt.plot(nni, label='NNI', marker='o', linestyle='-', color='b')
plt.plot(sni, label='SNI', marker='x', linestyle='--', color='r')
plt.title('Расхождение NNI и SNI')
plt.legend()
plt.grid(True)
plt.show()
```

```
/Users/grindelf/Programming/JUPITER/venv/lib/python3.12/site-packages/matplo
tlib/cbook.py:1398: RuntimeWarning: invalid value encountered in cast
  return np.asarray(x, float)
```



Выводы

1. Без использования предложенного в статье оптимизатора обучения, нейросеть уже показывает неплохое значение функции потерь.
2. Следует уделить большее время подбору подходящего размера скрытого слоя и параметров обучения (по первому в статье присутсвтуют определенные рекомендации).

3. Следует выяснить, почему полученный по формуле (15) интеграл так сильно отличается от такого же, полученного методом Симпсона на широких (шире 10) интервалах.