export

March 19, 2022

1 Машинное обучение в Jupyter Notebook

Первым делом необходимо импортировать необходимые для обучения библиотеки

[1]:
```
import numpy as np # for matrix operations
from sklearn.neural_network import MLPRegressor as mlp # main ml class
from matplotlib import pyplot as plt # для построения графиков
# show plot inside notebook
%matplotlib inline
from IPython.display import set_matplotlib_formats #  hi-res plots
set_matplotlib_formats('pdf', 'png')
from joblib import dump, load # для загрузки и выгрузки моделей
plt.rcParams['figure.figsize'] = [10, 10]
```

```
/tmp/ipykernel_787542/2688301736.py:7: DeprecationWarning:
`set_matplotlib_formats` is deprecated since IPython 7.23, directly use
`matplotlib_inline.backend_inline.set_matplotlib_formats()`
  set_matplotlib_formats('pdf', 'png')
```

1.1 Добавление данных

У нас имеются экспериментальные данные полнофакторного эксперимента по измерению характеристик фотонного кристалла в зависимости от времени распыления магнетрона и мощности на нём. В качестве входных данных выступают два массива: мощность на магнетроне X1 и время распыления X2

Выходные параметры – ширина запрещённой зоны, нм; длина отражённой волны, нм; степень отражения, %

[2]:
```
#input
X0 = np.ones(4)
X1 = np.array([150, 150, 200, 200])
X2 = np.array([5, 10, 5, 10])

#output
Y1 = np.array([53.5, 50.5, 56.5, 38])
Y2 = np.array([571.5, 556.5, 575, 532.5])
Y3 = np.array([18.9, 8.185, 12.05, 5.775])
```

1.2 Нормализация данных

Для корректной работы нейросети данные неоходимо нормализровать в пределах $x \in [-1; 1]$

[3]:
```
X1 = np.interp(X1, (X1.min(), X1.max()), (-1, +1))
X2 = np.interp(X2, (X2.min(), X2.max()), (-1, +1))
```

1.3 Выбор оптимальных параметров для обучения

При обучении необходимо учитывать множество факторов, среди них:

- размерность сети;
- число скрытых слоёв;
- функция активации;
- оптимизатор.

1.3.1 Методы контроля

В качестве метода контроля за качеством обучения предлагается следующий алгоритм:

1. Взять заранее рассчитанные математические модели из (вставить предыдущий пункт);
2. Подставить в них некие случайные значения, то есть получить отображение $\mathbb{X} \to \mathbb{Y}$;
3. Использовать полученные массивы в качестве *test*-выборки для проверки качества полученной модели.

Создадим 4 варианта тестирований:

1. *X_matmod_test, y_matmod_test*, для удобства отображения предсказаний моделей с неизменными размерностями на входе и выходе;
2. *X_matmod_test_12, y_matmod_test*, для удобства отображения предсказаний модели с двумя входными векторами и тремя выходными;
3. *X_matmod_test_full, y_matmod_test_full*, для валидации моделей с неизменными размерностями на входе и выходе;
4. *X_matmod_test_notfull, y_matmod_test_full*, для валидации модели с двумя входными векторами и тремя выходными;

[4]:
```
# создадим 3 массива размерностью 1000 (входные данные для валидации)
X0_matmod = np.ones(1000)
X1_matmod = np.linspace(-1,1,1000)
X2_matmod = np.linspace(-1,1,1000)

# создадим массивы небольшой размерности для удобства сверки
X0_matmod_test = np.ones(4) # генерация 4 единиц в массив
X1_matmod_test = np.random.choice(X1_matmod,size = 4, replace =
 ↪False) # генерация 4 значений без повторений
X2_matmod_test = np.random.choice(X2_matmod,size = 4, replace = False)
```

```
X_matmod_test = np.transpose(np.array([X0_matmod_test,
 ↪X1_matmod_test, X2_matmod_test])) # сборка массива и
 ↪транспонирование
X_matmod_test_12 = np.transpose(np.array([X1_matmod_test,
 ↪X2_matmod_test])) # специальный массив без X0 для проверки
 ↪необходимости в X0

# генерация выходных данных из рассчитанной модели для теста
y1_matmod_test = 49.625 - 2.375 * X1_matmod_test - 5.375 *
 ↪X2_matmod_test - 3.875 * X1_matmod_test * X2_matmod_test #
 ↪вычисляем Y при помощи X
y2_matmod_test = 558.875 - 5.125 * X1_matmod_test - 14.375 *
 ↪X2_matmod_test - 6.875 * X1_matmod_test * X2_matmod_test
y3_matmod_test = 11.122 - 2.315 * X1_matmod_test - 4.2475 *
 ↪X2_matmod_test + 1.11 * X1_matmod_test * X2_matmod_test
y_matmod_test = np.transpose(np.array([y1_matmod_test,
 ↪y2_matmod_test, y3_matmod_test]))

# получаем на выходе три массива
print(X_matmod_test)
print(X_matmod_test_12)
print(y_matmod_test)

# генерируем большой массив для более точной сверки
X_matmod_test_full = np.transpose(np.array([X0_matmod, X1_matmod,
 ↪X2_matmod])) # сборка большого массива и транспонирование
X_matmod_test_notfull = np.transpose(np.array([X1_matmod,
 ↪X2_matmod])) # специальный массив без X0 для проверки
 ↪необходимости в X0
y1_matmod_test_full = 49.625 - 2.375 * X1_matmod - 5.375 * X2_matmod
 ↪- 3.875 * X1_matmod * X2_matmod # вычисляем Y при помощи X
y2_matmod_test_full = 558.875 - 5.125 * X1_matmod - 14.375 *
 ↪X2_matmod - 6.875 * X1_matmod * X2_matmod
y3_matmod_test_full = 11.122 - 2.315 * X1_matmod - 4.2475 * X2_matmod
 ↪+ 1.11 * X1_matmod * X2_matmod
y_matmod_test_full = np.transpose(np.array([y1_matmod_test_full,
 ↪y2_matmod_test_full, y3_matmod_test_full]))
```

```
[[ 1.          0.48348348  0.33533534]
 [ 1.         -0.20920921 -0.74174174]
 [ 1.          0.73773774 -0.36536537]
 [ 1.          0.04904905 -0.28328328]]
[[ 0.48348348  0.33533534]
 [-0.20920921 -0.74174174]
 [ 0.73773774 -0.36536537]
 [ 0.04904905 -0.28328328]]
[[ 46.04604905 550.46206417   8.7583622 ]
```

```
 [ 53.50741432 569.54287771  14.92911628]
 [ 50.88119401 562.19933497  10.66683289]
 [ 51.08499841 562.7913474   12.196274  ]]
```

1.4 Влияние размерности сети на качество модели

Создадим 2 массива $\mathbb{X}$: с 2 и 3 элементами соответственно. Сделано это для потому, что нейронной сети очень сложно сделать отображение из меньшей размерности в большую. Вектор $X0$ является ни чем иным, как вектором-заглушкой, не вносящим изменения в модель, но искусственно раздувающий размерность.

[5]:
```
X_012_train = np.array([X0,X1,X2]) # вектор размерности 3
X_12_train = np.array([X1,X2]) # вектор размерности 2
X_012_train
```

[5]:
```
array([[ 1.,  1.,  1.,  1.],
       [-1., -1.,  1.,  1.],
       [-1.,  1., -1.,  1.]])
```

Создадим 1 массив $\mathbb{Y}$, поскольку нас интересует 3 выходных параметра. Размерность этого вектора мы варьировать не будем

[6]:
```
Y_train=np.array([Y1,Y2,Y3])
Y_train
```

[6]:
```
array([[ 53.5  ,  50.5  ,  56.5  ,  38.   ],
       [571.5  , 556.5  , 575.   , 532.5  ],
       [ 18.9  ,   8.185,  12.05 ,   5.775]])
```

1.4.1 Транспорирование массивов

Получившиеся выше массивы не совсем подходят для обработки. Нам необходимо отображение комбинаций $\mathbb{X}$ в комбинации $\mathbb{Y}$. Для этого транспонируем обе матрицы.

[7]:
```
X_012_train = np.transpose(X_012_train)
X_12_train = np.transpose(X_12_train)
X_012_train
```

[7]:
```
array([[ 1., -1., -1.],
       [ 1., -1.,  1.],
       [ 1.,  1., -1.],
       [ 1.,  1.,  1.]])
```

[8]:
```
Y_train = np.transpose(Y_train)
Y_train
```

```
[8]: array([[ 53.5  , 571.5  ,  18.9  ],
            [ 50.5  , 556.5  ,   8.185],
            [ 56.5  , 575.   ,  12.05 ],
            [ 38.   , 532.5  ,   5.775]])
```

1.4.2 Создание архитектуры модели

Создадим 2 модели, отвечающие за размерности векторов. Оставим все значения по умолчанию кроме входных параметров.

```
[9]: model_012 = mlp(random_state=1,max_iter=1000000)
     model_12 = mlp(random_state=1,max_iter=1000000)
```

Проведём обучение моделей на заранее подготовленных выборках

```
[10]: model_012.fit(X_012_train, Y_train)
      model_12.fit(X_12_train, Y_train)
```

```
[10]: MLPRegressor(max_iter=1000000, random_state=1)
```

Проведём проверку качества работы моделей при помощи заранее заготовленных тестов. Функция *score* – функция $\mathbb{R}^2$. Наилучшая возможная оценка – 1, и она может быть отрицательной (потому что модель может быть произвольно хуже). Постоянная модель, которая всегда предсказывает ожидаемое значение y, игнорируя входные характеристики, получит оценку 0.

$$R = \left(1 - \frac{u}{v}\right) \tag{1}$$

- u – остаточная сумма квадратов. Метод оценки разницы между данными и оценочной моделью. Чем меньше разница, тем лучше оценка. По существу измеряет разброс ошибок моделирования. Другими словами, он показывает, как изменение зависимой переменной в регрессионной модели не может быть объяснено с помощью модели. Как правило, более низкая остаточная сумма квадратов указывает на то, что регрессионная модель может лучше объяснить данные, в то время как более высокая остаточная сумма квадратов указывает на то, что модель плохо объясняет данные.

$$u = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2}$$

  - y_i – действительное значение;
  - $\hat{y}_i$ – значение, полученное моделью.

- v – общая сумма квадратов. Представляет собой отклонение значений зависимой переменной от выборочного среднего значения зависимой переменной. По сути, общая сумма квадратов количественно определяет общую вариацию в выборке. Его можно определить по следующей формуле:

$$v = \sum_{i=1}^{n} (y_i - \bar{y}_i)^2 \tag{3}$$

  - y_i – действительное значение;

– $\bar{y}_i$ – среднее значение.

[11]:
```
print("Модель 012")
print(model_012.score(X_matmod_test_full,y_matmod_test_full))
print(model_012.predict(X_matmod_test))

print("Модель 12")
print(model_12.score(X_matmod_test_notfull,y_matmod_test_full))
print(model_12.predict(X_matmod_test_12))
```

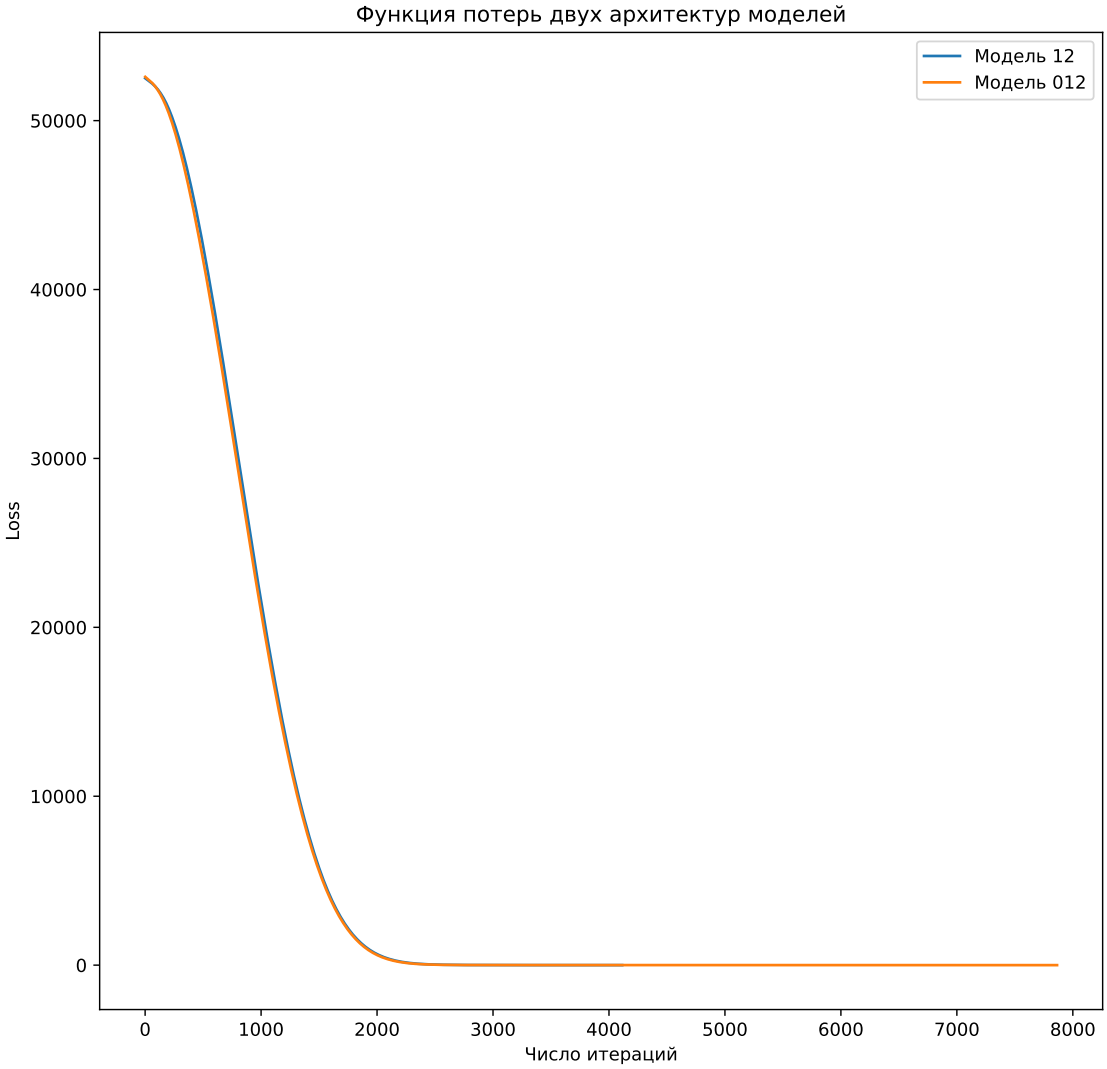
```
Модель 012
0.8135462211671235
[[ 43.40571923 543.68112196   9.65241689]
 [ 50.90158838 562.20556561  15.87171834]
 [ 46.43032235 552.39883876  12.27512113]
 [ 47.8070884  554.49364847  13.29834493]]
Модель 12
-16.94187285805916
[[ 36.32865331 440.61778227   5.53561853]
 [ 48.04352244 484.47184626  13.96828786]
 [ 41.28359463 459.47902463   8.88703558]
 [ 43.18840615 465.67895223  10.44771188]]
```

Как видно, модель с раздутой размерностью показала себя лучше. Построим график сходимости каждой из моделей

[12]:
```
y_curve_012 = model_012.loss_curve_
x_curve_012 = [i for i in range(1,len(y_curve_012)+1)]
print(f'Модель 012 сошлась за {len(x_curve_012)} итераций, потери
  ↪составили {model_012.best_loss_}')

y_curve_12 = model_12.loss_curve_
x_curve_12 = [i for i in range(1,len(y_curve_12)+1)]
print(f'Модель 12 сошлась за {len(x_curve_12)} итераций, потери
  ↪составили {model_12.best_loss_}')

plt.plot(x_curve_12, y_curve_12,label='Модель 12')
plt.plot(x_curve_012, y_curve_012,label='Модель 012')
plt.xlabel("Число итераций")
plt.ylabel("Loss")
plt.legend()
plt.title("Функция потерь двух архитектур моделей")
```

```
Модель 012 сошлась за 7864 итераций, потери составили 0.
  ↪05247663242357793
Модель 12 сошлась за 4114 итераций, потери составили 0.
  ↪03380976458075128
```

[12]: Text(0.5, 1.0, 'Функция потерь двух архитектур моделей')



1.5 Влияние числа скрытых слоёв сети на качество модели

Как мы убедились в прошлом разделе, модель должна иметь размерность на входе не ниже размерности на выходе, иначе потери R^2 слишком высоки. Посмотрим теперь что будет, если варьировать количество скрытых слоёв модели (например, от 1 до 100). Необходимо помнить, что слишком сложная модель может переобучиться на небольшой выборке. Будем сохранять модели в отдельные файлы, поскольку их обучение занимает длительное время

[ ]:
```
loss = []
points = []
iters = []
for layers in range(2,101):
```

```
    varymodel =
 ↪mlp(random_state=1,max_iter=1000000,hidden_layer_sizes=(layers))
    varymodel.fit(X_012_train, Y_train)
    loss.append(varymodel.best_loss_)
    points.append(varymodel.loss_curve_)
    iters.append(varymodel.n_iter_)
    #dump(varymodel,f'models/layers/{layers}.joblib')
    print(layers)
```

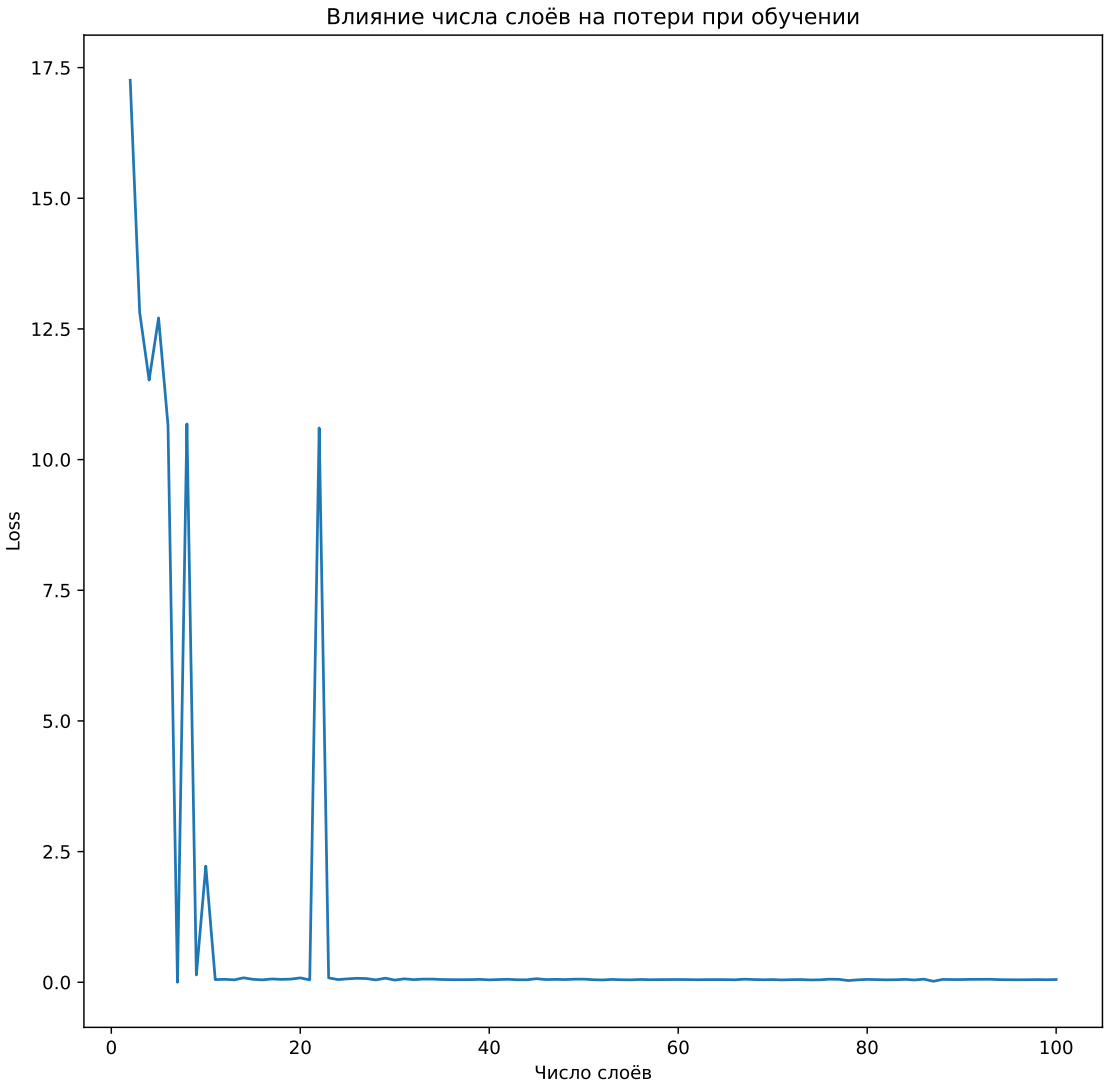
Для того, чтобы не проводить многократную тренировку модели, проведём тренировку один раз, а в остальные будем загружать модели из готовых файлов.

[13]:
```
loss = []
points = []
iters = []
scores = []
for layers in range(2,101):
    varymodel = load(f'models/layers/{layers}.joblib')
    loss.append(varymodel.best_loss_)
    points.append(varymodel.loss_curve_)
    iters.append(varymodel.n_iter_)
    scores.append(varymodel.
 ↪score(X_matmod_test_full,y_matmod_test_full))
```

[14]:
```
loss[5] = 0 # выборочная ошибка, занулим её
scores[5] = 0
```

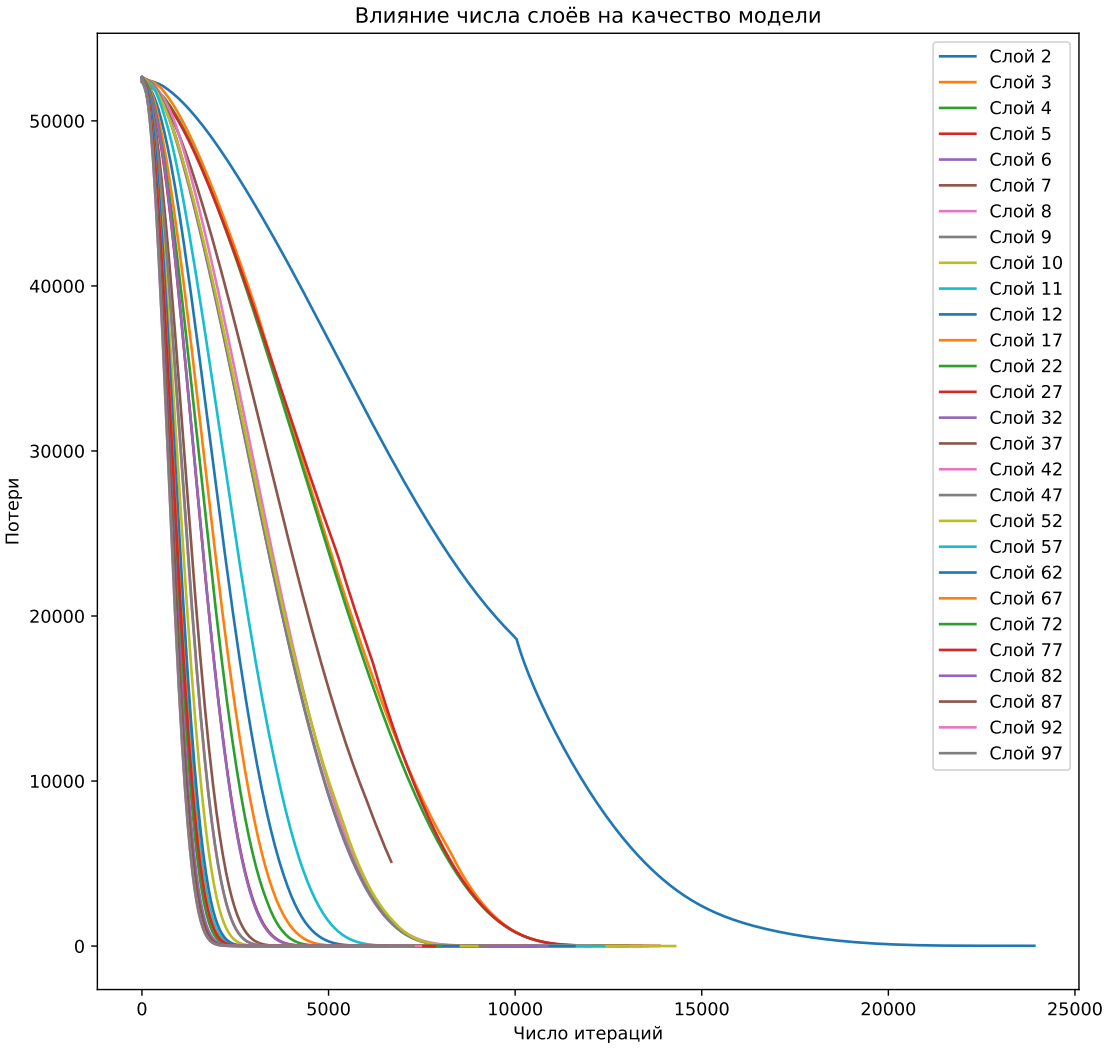
[15]:
```
layers = list(range(2,101))
plt.plot(layers, loss)
plt.xlabel("Число слоёв")
plt.ylabel("Loss")
plt.title("Влияние числа слоёв на потери при обучении")
```

[15]:
```
Text(0.5, 1.0, 'Влияние числа слоёв на потери при обучении')
```



[16]:
```
for i in range (len(iters)):
    #print(i)
    if (i>10 and i % 5 != 0):
        continue
    x = list(range(1,iters[i]+1))
    y = points[i]
    plt.plot(x, y, label=f'Слой {i+2}')
plt.xlabel("Число итераций")
plt.ylabel("Потери")
plt.title("Влияние числа слоёв на качество модели")
plt.legend()
```

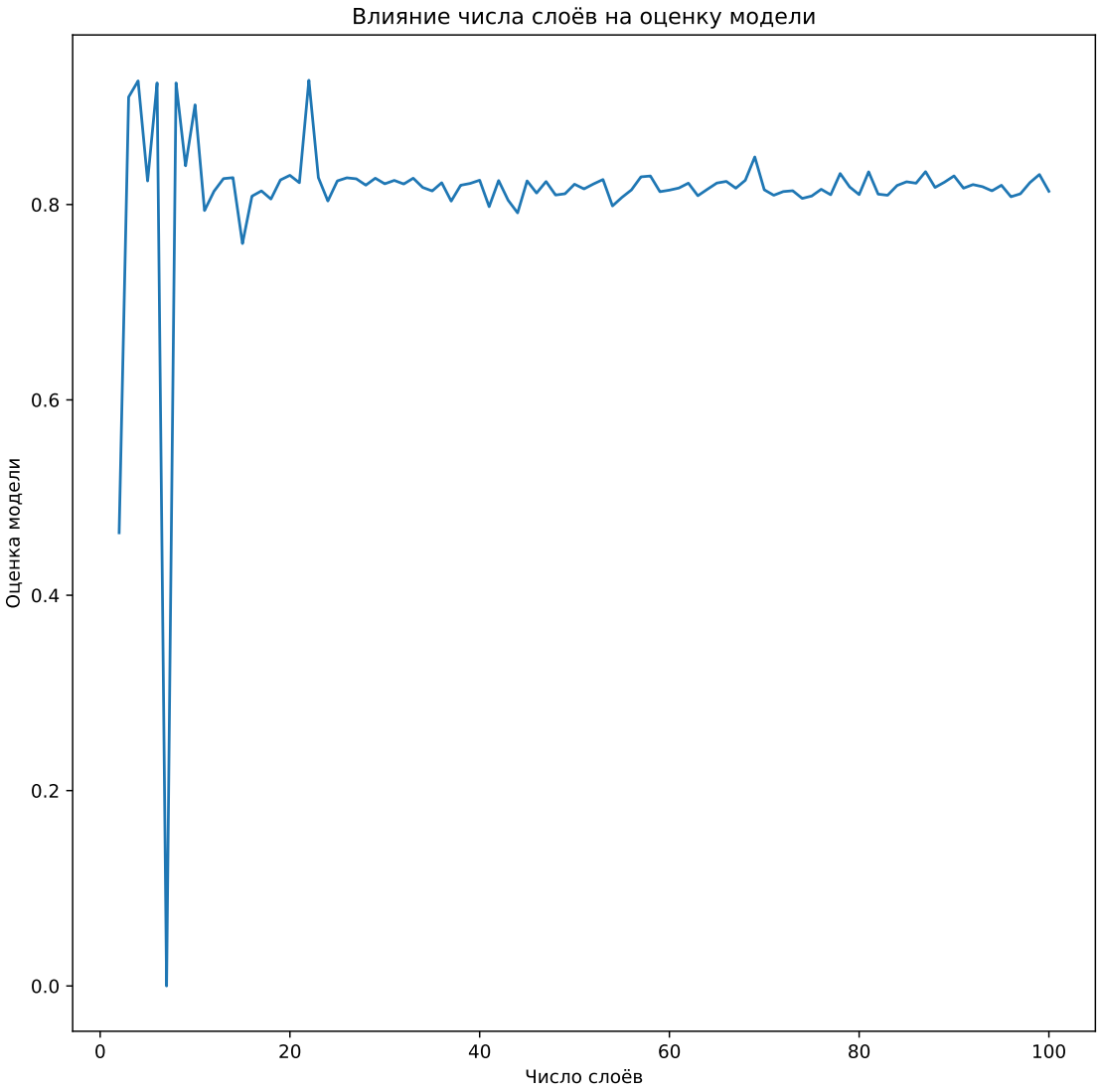
[16]: <matplotlib.legend.Legend at 0x7fe3db519d90>



Самым главным параметром при обучении будет являться её оценка. Проведём оценку модели

[17]:
```
plt.plot(list(range(2,101)), scores)
plt.xlabel("Число слоёв")
plt.ylabel("Оценка модели")
plt.title("Влияние числа слоёв на оценку модели")
```

[17]:
```
Text(0.5, 1.0, 'Влияние числа слоёв на оценку модели')
```

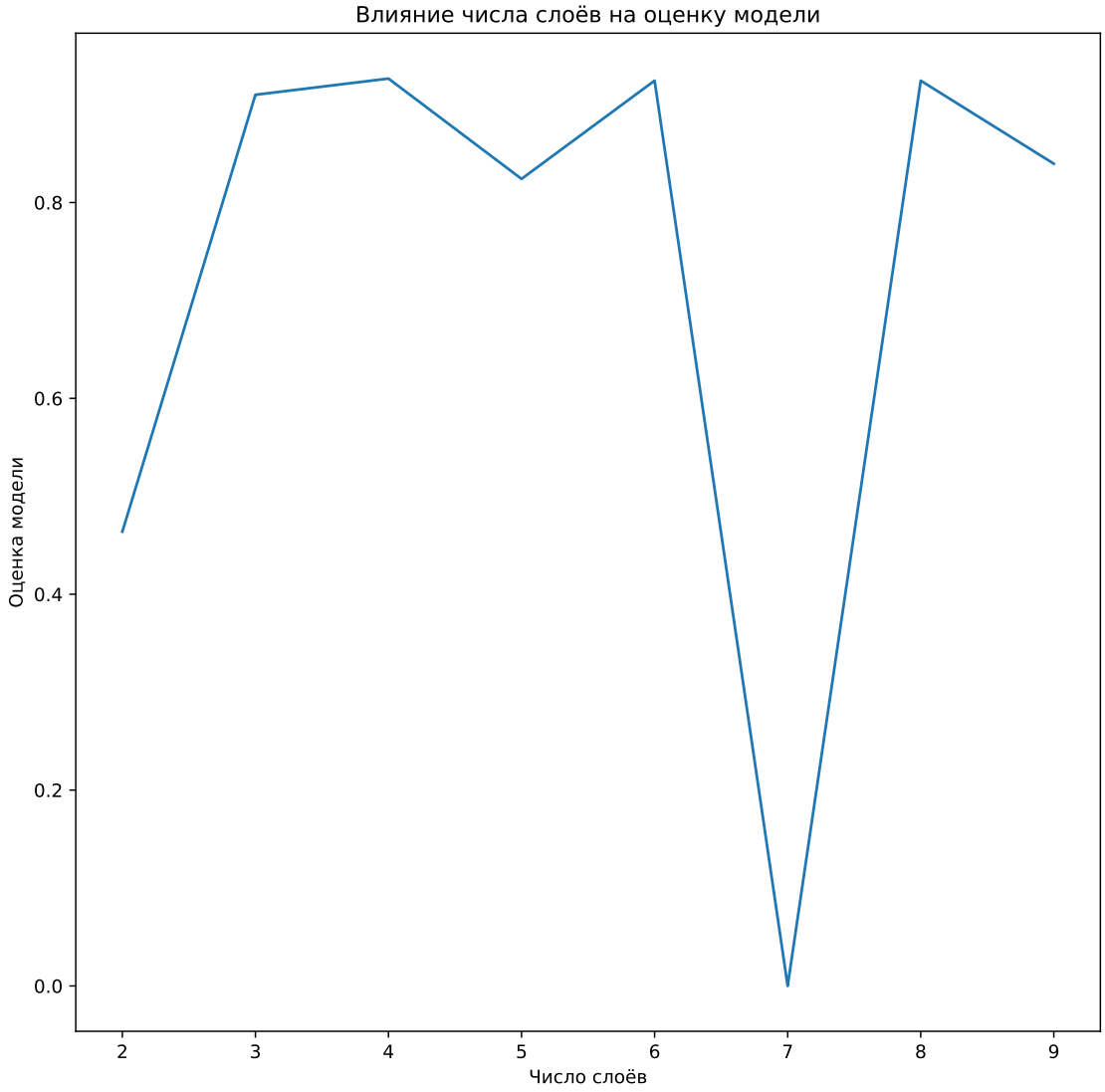


Очевидно, что нас интересует всё до десяти слоёв, дальше модель начинает переобучаться (резкое падение оценки – признак переобучения)

[18]:
```
plt.plot(list(range(2,10)), scores[:8])
plt.xlabel("Число слоёв")
plt.ylabel("Оценка модели")
plt.title("Влияние числа слоёв на оценку модели")
for i in range(len(scores[:8])):
    print(f'Число слоёв {i+2}, оценка модели {scores[i]}')
```

```
Число слоёв 2, оценка модели 0.4639444511330419
Число слоёв 3, оценка модели 0.9100311255101875
Число слоёв 4, оценка модели 0.9265799685143974
Число слоёв 5, оценка модели 0.8241257764470432
```

```
Число слоёв 6, оценка модели 0.924471640369974
Число слоёв 7, оценка модели 0
Число слоёв 8, оценка модели 0.9244552323152719
Число слоёв 9, оценка модели 0.8396396016693698
```



Из графика видно, что лучше всего нам подходит модель с числом слоёв, равным 8 (далее наблюдается переобучение)

1.6 Влияние функции активации на результаты модели

Библиотека *scikit-learn* предоставляет доступ к 4 функциям активации:

1. *identity* – функция тождества. Безоперационная активация, полезная для реализации линейного узкого места

$$f(x) = x \tag{4}$$

2. *logistic* – сигмоидальная функция

$$f(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

3. *tanh* – гиперболический тангенс

$$f(x) = \tanh = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{6}$$

4. *relu* – линейный выпрямитель

$$f(x) = \begin{cases} 0, x < 0 \\ x, x \geqslant 0 \end{cases} \tag{7}$$

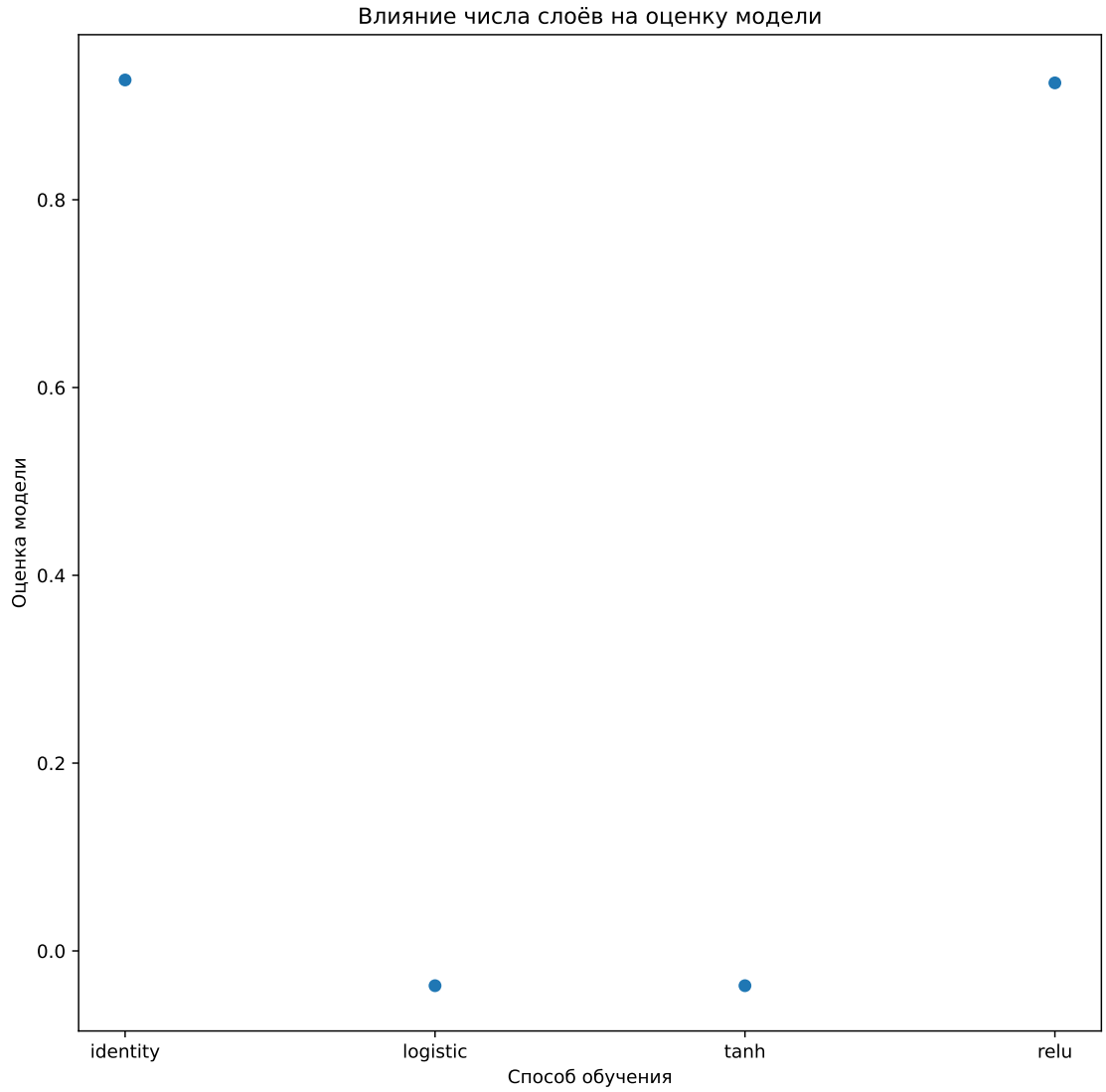
Обучим 4 модели при помощи каждой из функций активаций

[19]:
```
scores_activation = []
model_identity =
  ↪mlp(random_state=1,max_iter=2000000,hidden_layer_sizes=(8),\
                      activation='identity')
model_identity.fit(X_012_train, Y_train)
scores_activation.append(model_identity.
  ↪score(X_matmod_test_full,y_matmod_test_full))

model_logistic =
  ↪mlp(random_state=1,max_iter=2000000,hidden_layer_sizes=(8),\
                      activation='logistic')
model_logistic.fit(X_012_train, Y_train)
scores_activation.append(model_logistic.
  ↪score(X_matmod_test_full,y_matmod_test_full))

model_tanh =
  ↪mlp(random_state=1,max_iter=2000000,hidden_layer_sizes=(8),\
                  activation='tanh')
model_tanh.fit(X_012_train, Y_train)
scores_activation.append(model_tanh.
  ↪score(X_matmod_test_full,y_matmod_test_full))

model_relu =
  ↪mlp(random_state=1,max_iter=2000000,hidden_layer_sizes=(8),\
                  activation='relu')
model_relu.fit(X_012_train, Y_train)
scores_activation.append(model_relu.
  ↪score(X_matmod_test_full,y_matmod_test_full))
```

[20]:
```
x = [1, 2, 3, 4]
y = scores_activation
my_xticks = ['identity','logistic','tanh','relu']
plt.xticks(x, my_xticks)
```

```
plt.scatter(x,y)
plt.xlabel("Способ обучения")
plt.ylabel("Оценка модели")
plt.title("Влияние числа слоёв на оценку модели")
print(scores_activation)
```

```
[0.9275591949671171, -0.03702885533324888, -0.03702338056124536,
0.9244552323152719]
```



Из графика видно, что наилучший результат показывает отсутствие функции активации

1.7 Влияние солвера на результаты модели

Под солвером понимается один из алгоритмов для обучения нейросети. Библиотека *scikit-learn* предоставляет доступ к 3 алгоритмам обучения:

1. *lbfgs* – алгоритм Бройдена - Флетчера - Гольдфарба - Шанно, рекомендуется для небольших датасетов;
2. *sgd* – стохастический градиентный спуск, один из классических методов обучения;
3. *adam* – метод адаптивной скорости обучения, используется по умолчанию.

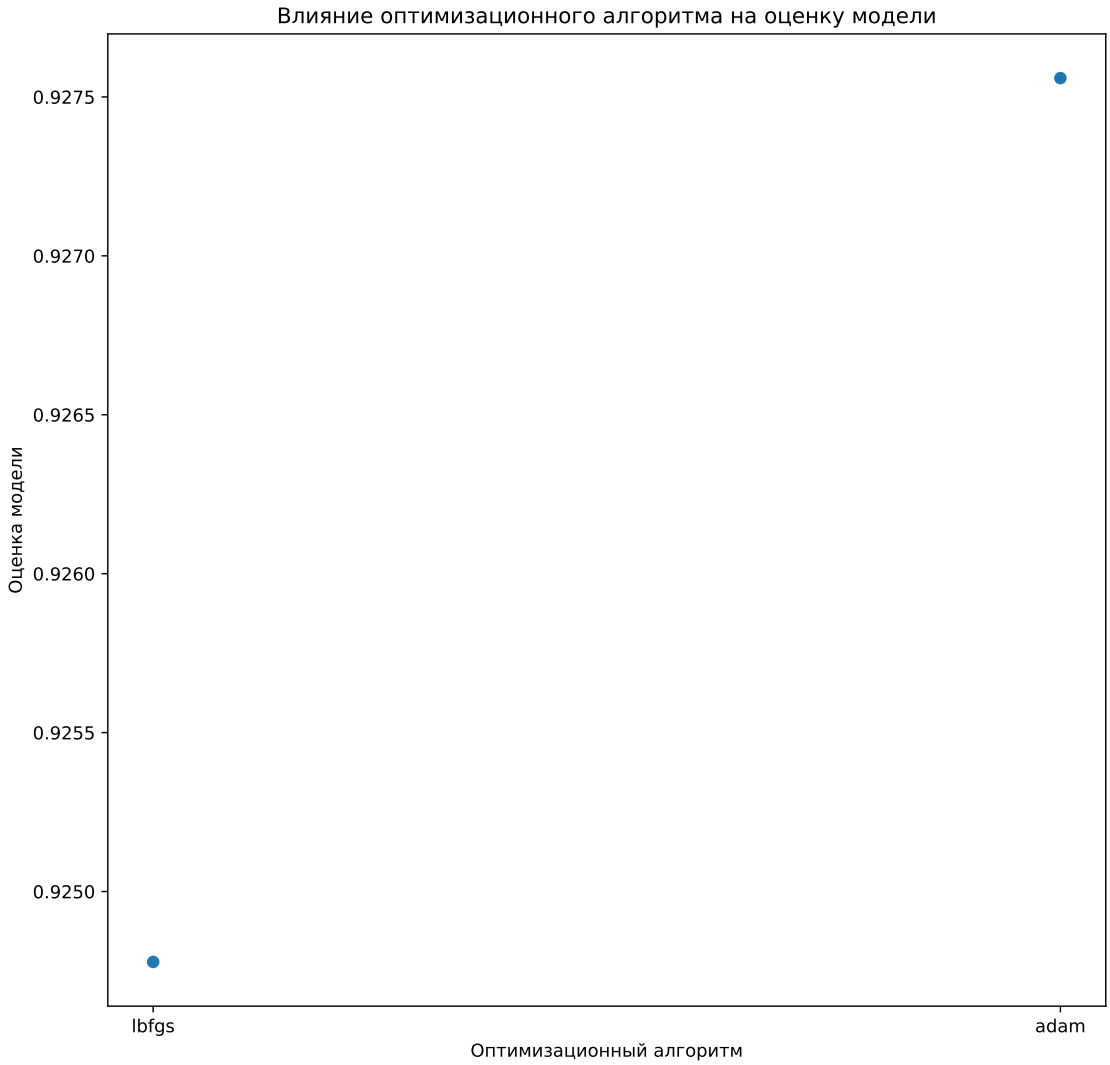
Обучим модель с использованием этих алгоритмов

[21]:
```
algorithms = []
model_lbfgs =
  ↪mlp(random_state=1,max_iter=2000000,hidden_layer_sizes=(8),\
                  activation='identity',solver='lbfgs')
model_lbfgs.fit(X_012_train, Y_train)
algorithms.append(model_lbfgs.
  ↪score(X_matmod_test_full,y_matmod_test_full))

model_adam =
  ↪mlp(random_state=1,max_iter=2000000,hidden_layer_sizes=(8),\
                 activation='identity',solver='adam')
model_adam.fit(X_012_train, Y_train)
algorithms.append(model_adam.
  ↪score(X_matmod_test_full,y_matmod_test_full))
```

При обучении модели методом стохастического градиентного спуска нейросеть не смогла добиться оптимальных параметров и обучение было аварийно завершено

[22]:
```
x = [1, 2]
y = algorithms
my_xticks = ['lbfgs','adam']
plt.xticks(x, my_xticks)
plt.scatter(x,y)
plt.xlabel("Оптимизационный алгоритм")
plt.ylabel("Оценка модели")
plt.title("Влияние оптимизационного алгоритма на оценку модели")
print(algorithms)
```

```
[0.9247783928754306, 0.9275591949671171]
```



1.8 Проверка полученной модели

После проведения экспериментов с моделями, попробуем проверить полученную модель на всей экспериментальной выборке. Обучим верную модель

[23]:
```
final_model =
  ↪mlp(random_state=1,max_iter=2000000,hidden_layer_sizes=(8),\
                  activation='identity',solver='adam')
final_model.fit(X_012_train, Y_train)
```

[23]:
```
MLPRegressor(activation='identity', hidden_layer_sizes=8,
  ↪max_iter=2000000,
             random_state=1)
```

[24]:
```
final_model.score(X_matmod_test_full,y_matmod_test_full)
```

[24]:
```
0.9275591949671171
```

Наша модель набрала 0.92 из 1, что является хорошим результатом при обучении на 4 входных данных

1.9 Получение результатов из модели

После обучения модели встаёт вопрос: как получать из неё результат. Для этого в библиотеке *scikit-learn* в классе *MLPRegressor* метод *predict*. Попробуем получить из модели какое-нибудь предсказание, например, для точки $X = [1, 0, 0]$, что соответствует времени напыления $t = 7.5$ минут при мощности $P = 175$ Вт

[25]:
```
[[a,b,c]] = final_model.predict([[1,0,0]])
print(f'''Данные по модели
Ширина запрещённой зоны {round(a,2)} нм
Длина отражённой волны {round(b,2)} нм
Процент отражённого света {round(c,2)} %''')
```

```
Данные по модели
Ширина запрещённой зоны 49.63 нм
Длина отражённой волны 558.62 нм
Процент отражённого света 11.23 %
```

Реальные данные (вычисленные по модели):

- Ширина запрещённой зоны 49.625 нм
- Длина отражённой волны 558.875 нм
- Процент отражённого света 11.122 %

Отклонения составили:

$$\delta S = \frac{S_{\text{real}} - S_{\text{model}}}{S_{\text{real}}} \cdot 100\% = \frac{49.625 - 49.63}{49.625} \cdot 100\% = -0.01\% \tag{8}$$

$$\delta \lambda = \frac{\lambda_{\text{real}} - \lambda_{\text{model}}}{\lambda_{\text{real}}} \cdot 100\% = \frac{558.675 - 558.62}{558.75} \cdot 100\% = 0.0098\% \tag{9}$$

$$\delta R = \frac{R_{\text{real}} - R_{\text{model}}}{R_{\text{real}}} \cdot 100\% = \frac{11.122 - 11.23}{11.122} \cdot 100\% = -0.97\% \tag{10}$$

Отклонения от матмодели получились меньше одного процента

[26]:
```
print(f'Предсказание модели {final_model.predict(X_matmod_test)}')
print(f'Реальные значения {y_matmod_test}')
```

```
Предсказание модели [[ 46.67454687 551.32429405   8.68390614]
 [ 54.10855985 570.35738101  14.86237024]
 [ 49.83670087 560.09384672  11.07154185]
 [ 51.03116084 562.44344424  12.31720136]]
Реальные значения [[ 46.04604905 550.46206417   8.7583622 ]
 [ 53.50741432 569.54287771  14.92911628]
 [ 50.88119401 562.19933497  10.66683289]
 [ 51.08499841 562.7913474   12.196274  ]]
```

Сохраним полученную модель

[27]:
```
dump(final_model,f'models/final_model.joblib')
```

[27]:
```
['models/final_model.joblib']
```