Лекция 9

Использование GPU при *глубоком обучении*.

- **Библиотеки *tensorflow* и *keras*.**

Тензоры в математике и физике

- **Тензор** - геометрический объект, представляемый в определённой системе координат индексированным набором чисел, меняющих свои значения по определённым правилам при преобразовании системы координат.
- **Ранг тензора** - количество индексов.
- Каждый индекс пробегает значения от 1 до N.
- N одинаково (!) для всех индексов и называется **размерностью** тензора.
- Правила преобразования тензора различны для двух типов индексов - **ковариантных** и **контравариантных**.
- **Валентность** определяет количество ковариантных и контрвариантных индексов.

Матричное представление тензоров

2-х валентный ковариантный тензор:

$$a_{ij} = \begin{pmatrix} a_{00} & a_{01} & a_{02} \\ a_{10} & a_{11} & a_{12} \\ a_{20} & a_{21} & a_{22} \end{pmatrix}$$

1-валентный ковариантный тензор:

$$b_k = \begin{pmatrix} b_0 & b_1 & b_2 \end{pmatrix}$$

1-валентный контравариантный тензор:

$$b^n = \begin{pmatrix} b^0 \\ b^1 \\ b^2 \end{pmatrix}$$

Тензорные операции

Сложение

$$c_{ij}^{k} = a_{ij}^{k} + b_{ij}^{k}$$

Умножение

$$Q_{ijkl}^{mn} = R_{ijkl} g^{mn}$$

$$c_{ij}^{k} = a_{ij} b^{k}$$

$$a_{ij} b^{k} = \begin{pmatrix} a_{00}b^{0}, a_{01}b^{0}, a_{10}b^{0}, ..., a_{22}b^{0}, \\ a_{00}b^{1}, a_{01}b^{1}, a_{10}b^{1}, ..., a_{22}b^{1}, \\ ..... \\ a_{00}b^{2}, a_{01}b^{2}, a_{10}b^{2} ..., a_{22}b^{2} \end{pmatrix}$$

Подстановка индексов

$$Q_{\mathbf{i}j\mathbf{k}l} = S_{\mathbf{k}j\mathbf{i}l}$$

Свёртка

$$d_i = c_{\mathbf{ij}}^{\mathbf{j}}$$

$$Q_{ij} = R_{i\mathbf{s}j\mathbf{p}} g^{\mathbf{sp}}$$

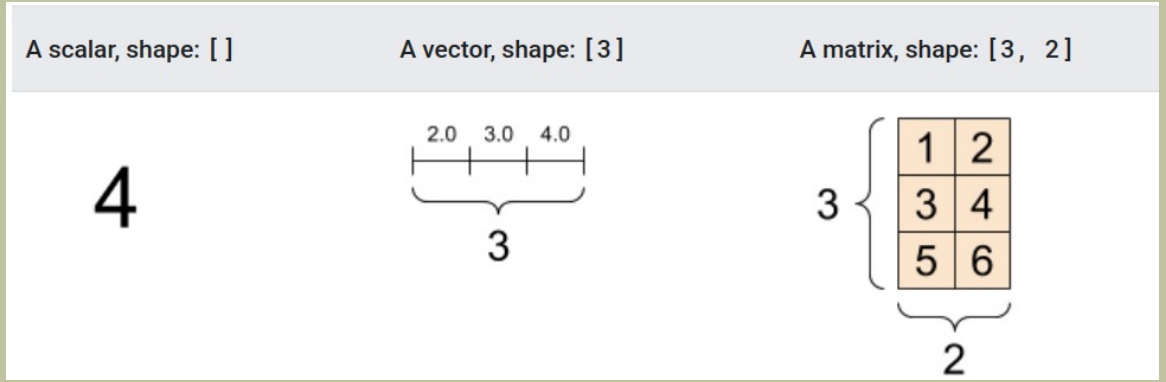
$$c_i = a_{i\mathbf{s}} b^{\mathbf{s}}$$

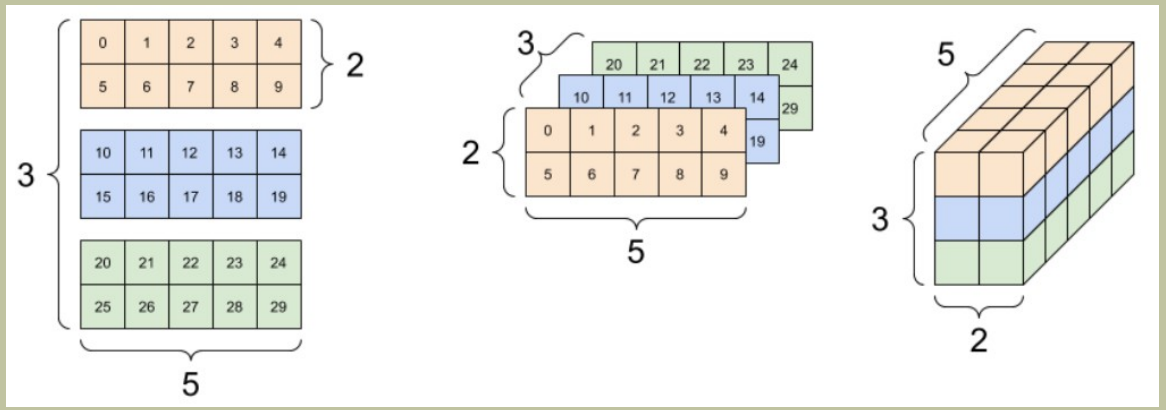
$$a_{ij} b^{j} = \left(a_{00}b^{0} + a_{01}b^{1} + a_{02}b^{2}, ..., a_{20}b^{0} + a_{21}b^{1} + a_{22}b^{2} \right)$$

Тензоры в программировании

Многомерные массивы ***numpy*** *и* ***tensorflow****:*

Shape (форма).
*Dimension***s** (размерност**и**)**.**





```
import numpy as np
a = np.array( [11, 42, 14, 75,32] )
print(a)

a = np.array([  [ [11, 22], [54, 65] ],
                [ [37, 71], [27, 29] ],
                        [ [31, 73], [65,
48] ]  ])
print(a)
```

Массивы
numpy

```
[11 42 14 75 32]

[  [ [11 22]
     [54 65] ]
   [ [37 71]
     [27 29] ]
   [ [31 73]
     [65 48] ]  ]
```

```
import tensorflow as tf
a = tf.constant( [11, 42, 14, 75,32] )
print(a)
a =  tf.constant( [  [ [11, 22], [54, 65] ],
                     [ [37, 71], [27, 29] ],
                          [ [31, 73], [65,
48] ]  ])
print(a)
```

```
print(a.numpy())
```

```
[  [ [11 22]
     [54 65] ]
   [ [37 71]
     [27 29] ]
   [ [31 73]
     [65 48] ]
]
```

```
tf.Tensor([11 42 14 75 32],
          shape=(5,), dtype=int32)
tf.Tensor(
[  [ [11 22]
     [54 65] ]
   [ [37 71]
     [27 29] ]
   [ [31 73]
     [65 48] ]
], shape=(3, 2, 2), dtype=int32)
```

Доступ к элементам тензора. Слайсы.

```
b=a[1:3,0:2,0:2]
print(b)
```

```
tf.Tensor(
[  [ [37 71]
     [27 29] ]
   [ [31 73]
     [65 48] ]  ],
shape=(2, 2, 2), dtype=int32)
```

```
b=a[1:3,0:2,0:2]
print(b[1,0,0:2])
```

```
tf.Tensor([31 73], shape=(2,), dtype=int32)
```

```
a = tf.constant([ [1.0, 2],
                  [3, 4],
                  [5, 6] ])
d=a[0:3,0]
print(d)
dt=tf.reshape(d,(3,1))
print(dt)

e=tf.constant([[1],[2],[3.5]])
print(e)
```

```
tf.Tensor([1. 3. 5.], shape=(3,), dtype=float32)

tf.Tensor(
[[1.]
 [3.]
 [5.]], shape=(3, 1), dtype=float32)

tf.Tensor(
[[1. ]
 [2. ]
 [3.5]], shape=(3, 1), dtype=float32)
```

“Тензорные” операции

```
print(tf.transpose(dt)@e)
print(tf.matmul(dt,e,transpose_a=True))

print(tf.transpose(dt)*e)
print(dt*e)
print(tf.multiply(dt, e))

b = tf.constant([[2.0, 2],
                 [10, 3],
                 [4, 6]])
print(tf.matmul(a,b,transpose_a=True))
```

```
tf.Tensor([[24.5]], shape=(1, 1), dtype=float32)
tf.Tensor([[24.5]], shape=(1, 1), dtype=float32)
tf.Tensor(
[[ 1.   3.   5. ]
 [ 2.   6.  10. ]
 [ 3.5 10.5 17.5]], shape=(3, 3), dtype=float32)
tf.Tensor(
[[ 1. ]
 [ 6. ]
 [17.5]], shape=(3, 1), dtype=float32)
tf.Tensor(
[[ 1. ]
 [ 6. ]
 [17.5]], shape=(3, 1), dtype=float32)
tf.Tensor(
[[52. 41.]
 [68. 52.]], shape=(2, 2), dtype=float32)
```

Modul(*tf.Module*) - контейнер для Variables

tf-s-models.py

```
import tensorflow as tf

class Dense(tf.Module):  #наследует tf.Module
  def  __init__(self, in_features, out_features, name=None): #конструктор
    super().__init__(name=name)
    self.w = tf.Variable(
            tf.random.normal([in_features, out_features]), name='w')
    self.b = tf.Variable(tf.zeros([out_features]), name='b')
  def __call__(self, x):  #функтор
    y = tf.matmul(x, self.w) + self.b
    return tf.nn.relu(y) #relu(y)=max(0,y)
```

```
>>> d=Dense(in_features=3, out_features=2)
>>> d.w
<tf.Variable 'w:0' shape=(3, 2) dtype=float32, numpy=
array([[ 0.7998288 ,  0.05016818],
                [ 0.9616611 , -1.6000011 ],
          [ 0.8098894 ,  0.69114935]], dtype=float32)>
>>> d.b
<tf.Variable 'b:0' shape=(2,) dtype=float32, numpy=array([0., 0.], dtype=float32)>
>>> d(tf.ones([1,3]))
<tf.Tensor: shape=(1, 2), dtype=float32, numpy=array([[2.5713792, 0.       ]],
dtype=float32)>
>>> tf.ones([1,3])@d.w
<tf.Tensor: shape=(1, 2), dtype=float32, numpy=array([[ 2.5713792, -0.8586836]],
dtype=float32)>
```

```
class SequentialModule(tf.Module):
  def __init__(self, name=None):
    super().__init__(name=name)

    self.dense_1 = Dense(in_features=3, out_features=3)
    self.dense_2 = Dense(in_features=3, out_features=2)

  def __call__(self, x):
    x = self.dense_1(x)
    return self.dense_2(x)

# Создание модели
my_model = SequentialModule(name="the_model")

# Вызов со случайными параметрами
print("Model results:", my_model(tf.constant([[2.0, 2.0, 2.0]])))
```

Этапы разработки

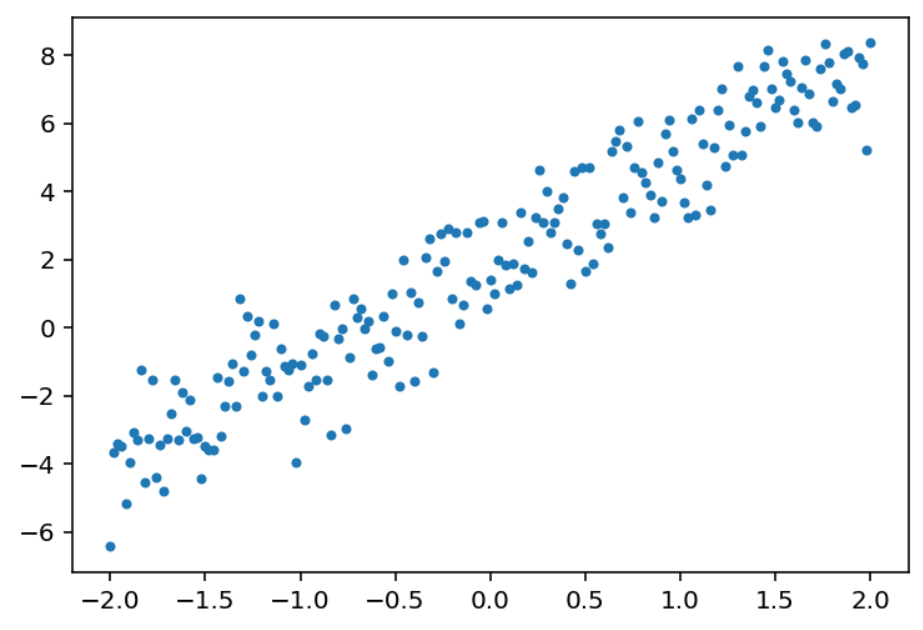
- Получение данных для обучения.
- Задание модели.
- Задание функции потерь.
- Прямой проход по обучающим данным, вычисление потерь при сравнении с идеальными выходными значениями.
- Использовать оптимизатор для настройки переменных с целью уменьшения потерь.
- Оценка результата.

Пример: линейная регрессия (tensorflow)

train_loop.py

```
import tensorflow as tf
import matplotlib.pyplot as plt
colors = plt.rcParams['axes.prop_cycle'].by_key()['color']

TRUE_W = 3.0
TRUE_B = 2.0
NUM_EXAMPLES = 201

x = tf.linspace(-2,2, NUM_EXAMPLES)
x = tf.cast(x, tf.float32)
```

```
def f(x):
  return x * TRUE_W + TRUE_B

noise = tf.random.normal(shape=[NUM_EXAMPLES])

y = f(x) + noise

plt.plot(x, y, '.')
plt.show()
```



```
class MyModel(tf.Module):
  def __init__(self, **kwargs):
    super().__init__(**kwargs)
    self.w = tf.Variable(5.0)
    self.b = tf.Variable(0.0)

  def __call__(self, x):
    return self.w * x + self.b
```

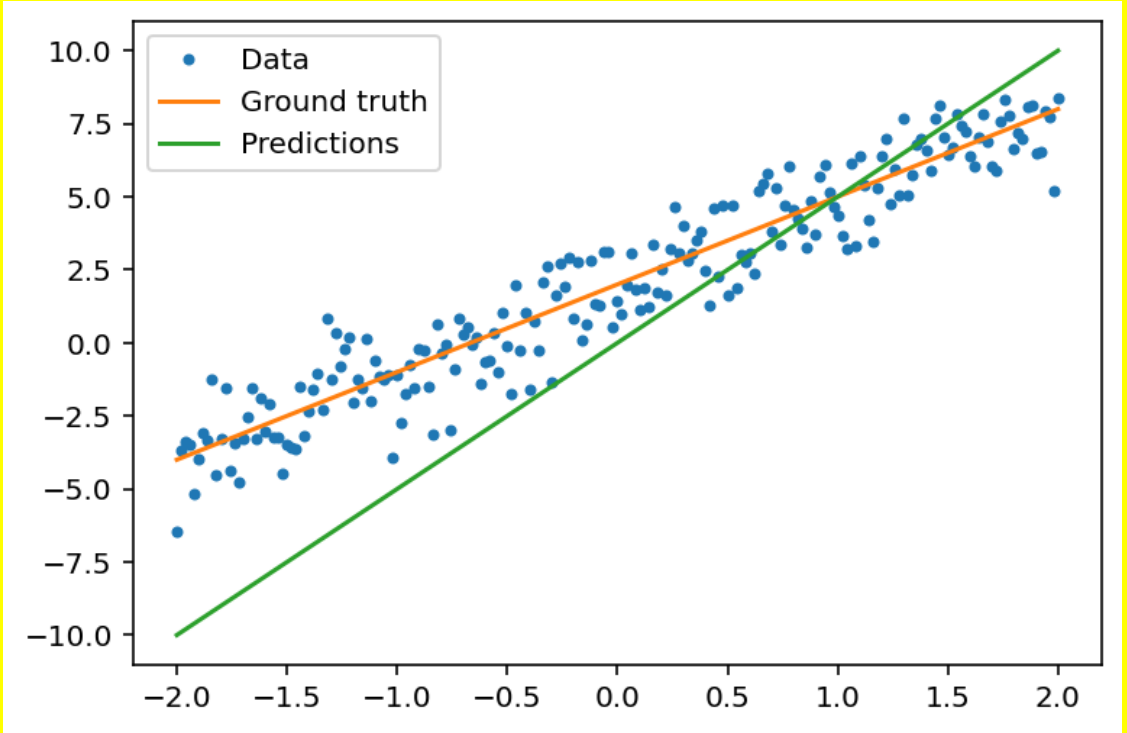
```
model = MyModel()
assert   model(3.0).numpy() == 15.0


print("Variables:", model.variables)
```

```
 Variables: (<tf.Variable 'Variable:0' shape=()
dtype=float32, numpy=0.0>, <tf.Variable
'Variable:0' shape=() dtype=float32, numpy=5.0>)
```

```
def loss(target_y, predicted_y):
  return tf.reduce_mean(tf.square(target_y - predicted_y))
```

```
plt.plot(x, y, '.', label="Data")
plt.plot(x, f(x), label="Ground truth")
plt.plot(x, model(x), label="Predictions")
plt.legend()
plt.show()
print("Current loss: %1.6f" % \
        loss(y, model(x)).numpy())
```

Current loss: **10.328803**



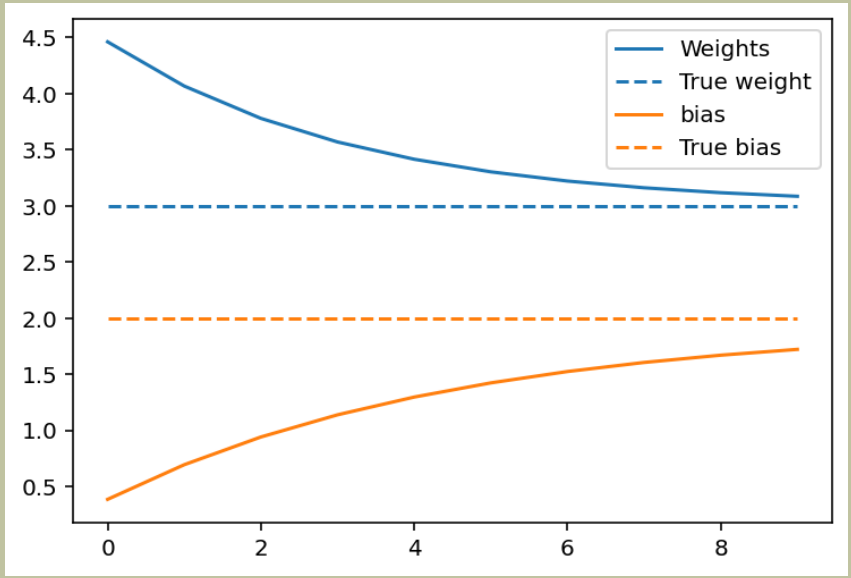
```
def train(model, x, y, learning_rate):
  with tf.GradientTape() as t:
    current_loss = loss(y, model(x))

  dw, db = t.gradient(current_loss, [model.w, model.b])
  model.w.assign_sub(learning_rate * dw)
  model.b.assign_sub(learning_rate * db)
```

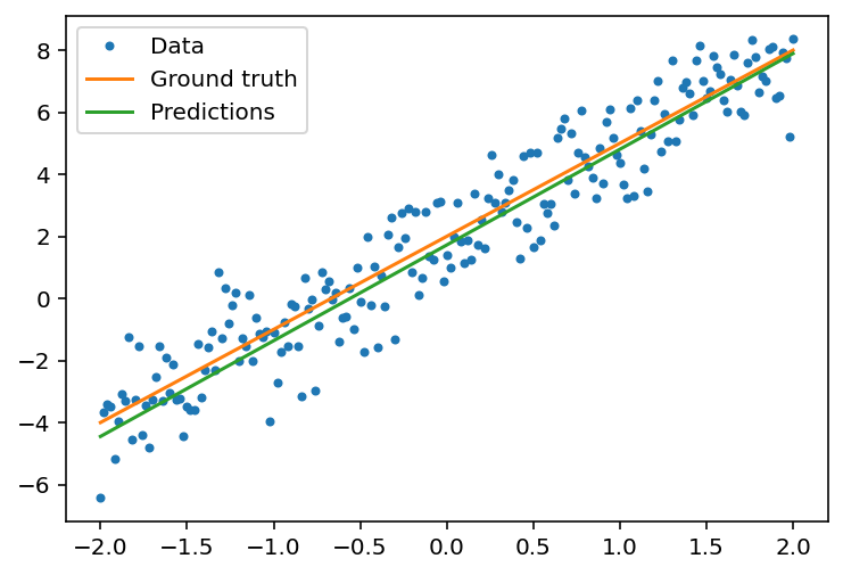
```
weights = []
biases = []
epochs = range(10)
def report(model, loss):
  return f"W = {model.w.numpy():1.2f}, b = {model.b.numpy():1.2f}, loss={loss:2.5f}"

def training_loop(model, x, y):
  for epoch in epochs:
    train(model, x, y, learning_rate=0.1)
    weights.append(model.w.numpy())
    biases.append(model.b.numpy())
    current_loss = loss(y, model(x))

    print(f"Epoch {epoch:2d}:")
    print("    ", report(model, current_loss))
```

```
current_loss = loss(y, model(x))

print("Starting:")
print("    ", report(model, current_loss))

training_loop(model, x, y)
```

```
Starting:
    W = 5.00, b = 0.00, loss=10.32880
Epoch  0:
    W = 4.46, b = 0.39, loss=6.47470
Epoch  1:
    W = 4.07, b = 0.69, loss=4.27498
……..
Epoch  8:
    W = 3.12, b = 1.67, loss=1.29955
Epoch  9:
    W = 3.09, b = 1.72, loss=1.26657
```

```
plt.plot(epochs, weights, label='Weights', color=colors[0])
plt.plot(epochs, [TRUE_W] * len(epochs), '--',
         label = "True weight", color=colors[0])

plt.plot(epochs, biases, label='bias', color=colors[1])
plt.plot(epochs, [TRUE_B] * len(epochs), "--",
         label="True bias", color=colors[1])

plt.legend()
plt.show()

plt.plot(x, y, '.', label="Data")
plt.plot(x, f(x), label="Ground truth")
plt.plot(x, model(x), label="Predictions")
plt.legend()
plt.show()
```

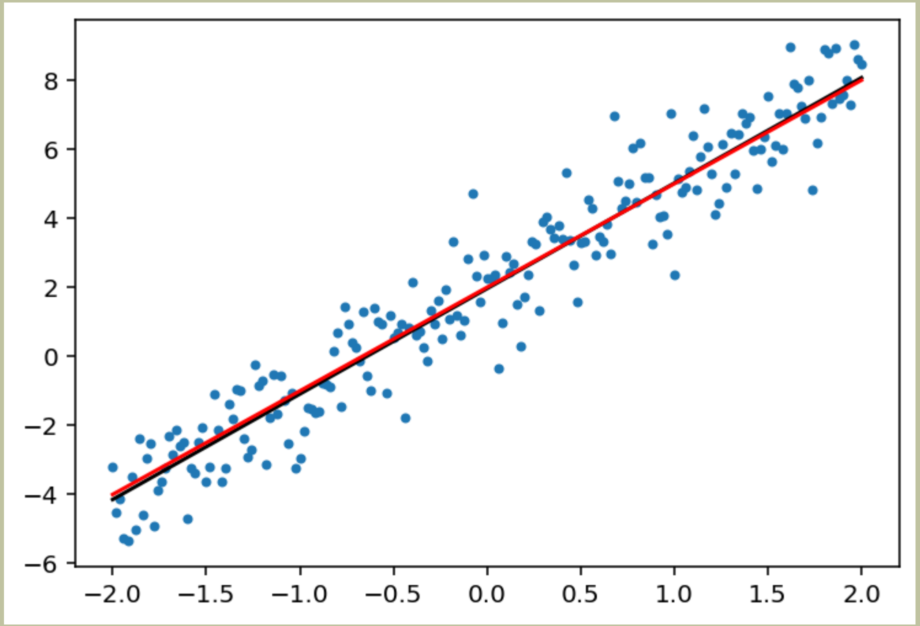

Weights
True weight
bias
True bias
4.5
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0
2
4
6
8


Data
Ground truth
Predictions
8
6
4
2
0
−2
−4
−6
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0

Пример: линейная регрессия (keras)

keras-regr-my.py

```
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()
model.add(Dense(units=1, input_shape=(1,)))
model.summary()

model.compile(optimizer=tf.keras.optimizers.SGD(learning_rate=0.05),
              loss='mse')
history=model.fit(x, y, epochs=20 )

y = model.predict(x)
plt.plot(x, y, color='black', label='Model Predictions')
plt.plot(x, f(x), color='red', label='True curve')
plt.show()
```



Epoch 1/20
7/7 ━━ 0s 6ms/step - loss: **10.9441**
Epoch 2/20
7/7 ━━ 0s 811us/step - loss: **2.7223**
Epoch 3/20
7/7 ━━ 0s 735us/step - loss: **1.3325**

,,,

Epoch 19/20
7/7━━ 0s 956us/step - loss: **1.0176**
Epoch 20/20
7/7━━ 0s 723us/step - loss: **1.0076**

Пример: распознавание цифр (tensorflow-keras)

https://www.tensorflow.org/datasets/keras_example

http://yann.lecun.com/exdb/mnist

keras-mnist.py

```
import tensorflow as tf
import tensorflow_datasets as tfds

(ds_train, ds_test), ds_info = tfds.load('mnist',
    split=['train', 'test'],
    shuffle_files=True,
    as_supervised=True,
    with_info=True,
)
```

```
def normalize_img(image, label):
    return tf.cast(image, tf.float32) / 255., label

ds_train = ds_train.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_train = ds_train.cache()
ds_train = ds_train.shuffle(ds_info.splits['train'].num_examples)
ds_train = ds_train.batch(128)
ds_train = ds_train.prefetch(tf.data.AUTOTUNE)

ds_test = ds_test.map(
    normalize_img, num_parallel_calls=tf.data.AUTOTUNE)
ds_test = ds_test.batch(128)
ds_test = ds_test.cache()
ds_test = ds_test.prefetch(tf.data.AUTOTUNE)
```

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dense(10)
])
model.compile(
    optimizer=tf.keras.optimizers.Adam(0.001),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
    metrics=[tf.keras.metrics.SparseCategoricalAccuracy()],
)

model.fit(
    ds_train,
    epochs=2,
    validation_data=ds_test,
)
```

Epoch 1/10
469/469 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3s 2ms/step - loss: **0.6337** -
sparse_categorical_accuracy: 0.8203 - val_loss: 0.1982 - val_sparse_categorical_accuracy: 0.9439
Epoch 2/10
469/469 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 874us/step - loss: 0.1778 -
sparse_categorical_accuracy: 0.9502 - val_loss: 0.1395 - val_sparse_categorical_accuracy: 0.9604
Epoch 3/10
469/469 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 867us/step - loss: 0.1241 -
sparse_categorical_accuracy: 0.9638 - val_loss: 0.1091 - val_sparse_categorical_accuracy: 0.9671
Epoch 4/10
469/469 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 862us/step - loss: 0.0934 -
sparse_categorical_accuracy: 0.9725 - val_loss: 0.0994 - val_sparse_categorical_accuracy: 0.9690
Epoch 5/10
469/469 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 871us/step - loss: 0.0734 -
sparse_categorical_accuracy: 0.9796 - val_loss: 0.0877 - val_sparse_categorical_accuracy: 0.9731
—--
Epoch 9/10
469/469 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 871us/step - loss: 0.0325 -
sparse_categorical_accuracy: 0.9906 - val_loss: 0.0722 - val_sparse_categorical_accuracy: 0.9773
Epoch 10/10
469/469 ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0s 875us/step - loss: 0**.0270** -
sparse_categorical_accuracy: 0.9933 - val_loss: 0.0746 - val_sparse_categorical_accuracy: 0.9774