УДК 519.68:007.5

Лабораторная работа 1. Многослойный персептрон

Цель работы: получить начальные знания о библиотеке Keras на примере простейших моделей персептронов.

§ 1. Модель из одного нейрона для линейной регрессии

1.1. Подготовка к работе. Для начала нужно убедиться, что у вас был полностью установлен дистрибутив Python с библиотекой tensorflow. Набьем следующие команды в рабочей тетради jupyter и проверим, что они исполнились без ошибок:

```
import pandas as pd
import tensorflow as tf
from matplotlib import pyplot as plt
```

В случае если часть библиотек отсутсвует следует их доустановить. Как вариант, можно вместо локально установленного Python'а использовать облачный сервис Google Colab с его реализацией тетради jupyter.

1.2. Построение модели. Подготовим функцию для построения модели:

```
def build_model(rate):
  # Строим модель в виде последовательно подключаемых
  # слоев (промежуточные слои соединяются автоматом).
  model = tf.keras.models.Sequential()
  # В нашем случае в виде одного слоя из
  # одного единственного нейрона:
  model.add(tf.keras.layers.Dense(units=1, # количество нейронов
                                  input_shape=(1,))) # размерность входов
  # Скомпилируем нашу модель в код, который исполнит
  # TensorFlow. В качестве функции потерь, которую
  # мы будем минимизировать, выберем среднеквадратическую ошибку
  model.compile(optimizer=tf.keras.optimizers.experimental.RMSprop(
                                                    learning_rate=rate),
                loss="mean_squared_error",
                metrics=[tf.keras.metrics.RootMeanSquaredError()])

  return model
```

Отдельно введем функцию для обучения модели на эталонных данных

```
def train_model(model, feature, label, epochs, batch_size):
  history = model.fit(x=feature, # входные данные  ("признаки")
                      y=label,   # выходные дынные ("метки")
                      batch_size=batch_size, # размер пакета
                      # данных для одной эпохи градиетного спуска
                      epochs=epochs) # общее число эпох обучения
```

```
  # Сохраняем итоговые веса нейрона в отдельные переменные
  trained_weight = model.get_weights()[0]
  trained_bias = model.get_weights()[1]
  # Отдельно сохраним список эпох обучения
  epochs = history.epoch
  # Преобразуем данные к формату Pandas.DataFrame
  hist = pd.DataFrame(history.history)
  # Сохраним данные о среднеквадратической ошибке
  rmse = hist["root_mean_squared_error"]
  # ---------
  return trained_weight, trained_bias, epochs, rmse
```

1.3. Визуализация результатов обучения.

```
def plot_the_model(trained_weight, trained_bias, feature, label):

  plt.xlabel("feature")
  plt.ylabel("label")
  # входные данные по оси x, выходные по оси y
  plt.scatter(feature, label)

  # Модель линейной регрессии будет представлена красной линией
  # с началом в  (x0, y0) и концом в (x1, y1).
  x0 = 0
  y0 = trained_bias
  x1 = feature[-1]
  y1 = trained_bias + (trained_weight * x1)
  plt.plot([x0, x1], [y0, y1], c='r')

  plt.show()
```

Отдельно определим функцию для визулацизации процесса обучения:

```
def plot_the_loss_curve(epochs, rmse):
  plt.figure()
  plt.xlabel("Epoch")
  plt.ylabel("Root Mean Squared Error")
  plt.plot(epochs, rmse, label="Loss")
  plt.legend()
  plt.ylim([rmse.min()*0.97, rmse.max()])
  plt.show()
```

1.4. Подготовка датасета и обучение. Подготовим искусственные данные для линейной регрессии:

```
features=([1.0,2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0,10.0,11.0,12.0])
labels=([5.0,8.8, 9.6,14.2,18.8,19.5,21.4,26.8,28.9,32.0,33.8,38.2])
```

Зададим основные параметры для обучения и проанализируем результаты.

```
learning_rate=0.01 # выберем малую скорость обучения
epochs=10 # возьмем небольшое число эпох для обучения
my_batch_size=12 # задействуем все обучающие примеры за одну эпоху обучения.
```

```
my_model = build_model(learning_rate)
trained_weight, trained_bias, epochs, rmse = train_model(my_model, features,
                                                         labels, epochs,
                                                         my_batch_size)
plot_the_model(trained_weight, trained_bias, my_feature, my_label)
plot_the_loss_curve(epochs, rmse)
```

Задание 1: Проанализировать с какими проблемами сталкивается наша модель при обучении.

Задание 2: Подобрать оптимальные параметры для обучения модели.

§ 2. Модель многослойного персептрона

2.1. Описание модели.

Определение 1. *Персептрон* — это многослойная нейронная сеть, состоящая из однотипных слоев $R_1, ..., R_n$, для которых используются:

- произвольная функция активации f (например $\sigma(x) = 1/(1 + e^{-x})$).
- для всех нейронов $r_{ij} \in R_j$ значения $y^{r_{ij}}(t) = f\left(\sum \omega_k^{r_{ij}} \cdot x_k^{r_{ij}}(t) - \theta^{r_{ij}}\right)$.
- сквозное подключение слоев $x_k^{r_{ij}}(t) = y^{r_{kj-1}}(t)$ для всех $j > 1$.
- общий параметр $\nu < 1$, задающий скорость обучения слоев.
- для выходного слоя R_n оценка ошибки (e_i — эталоны для выходов):

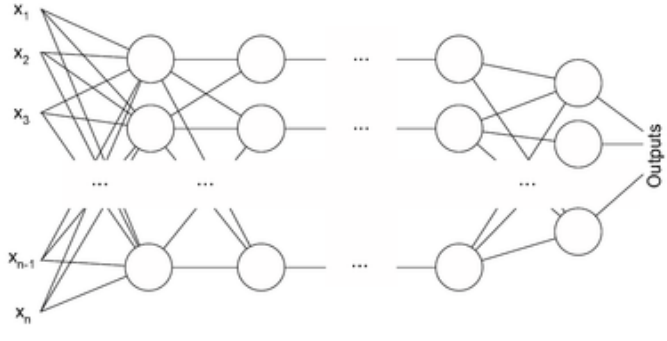

Рис. 1. Персептрон Румельхарта общего вида.

В библиотеке Keras для реализации персептронов используются полносвязные слои Dense в составе стека слоев Sequential:

```
model = tf.keras.models.Sequential()
model.add(tf.keras.Input(shape=(16,))) # размерность входных данных
model.add(tf.keras.layers.Dense(64,    # число нейроно в первом слое
            activation='relu'))        # активационная функция
model.add(tf.keras.layers.Dense(2))    # число нейроно в выходном слое
# по умолчанию activation='None'
```

2.2. Приложение полносвязных сетей для задачи классификации. Для решения задачи классификации наиболее часто используется схема с позиционным кодированием классов: для n классов мы используем ровно n выходных нейронов и активацию softmax. Отдельный нейрон в этом случае будет выдавать вероятность принадлежности входных данных к соответсвующему классу. Для обучения подобной модели более подходящей функцией потерь вместо среднеквадратической ошибки будет категоральная кроссэнтропия (categorical cross entropy). Рассмотрим работу с такими моделями на простых примерах:

Задание 3: Выбрать на двумерном пространстве несколько областей, разграниченнных прямыми линиями. Например, внутренняя или внешняя часть треугольника (квадрата, пятиугольника), либо некоторое количество полубесконечных секторов. Выбрать эталонные данные, которые будут отнесены к каждому из ваших классов. Перекодировать их номера в позиционные коды с помощью функции to_categorical.

Задание 4: Подготовить модель для обучения на выбранных данных. Для входного слоя число входов должно соотвествовать числу признаков (в нашем случае их 2), а число нейронов минимальному числу линий, которые необходимы для разграничения ваших классов на плоскости. В качестве активационной функции использовать любую нелинейную (например relu). Для выходного слоя использовать активацию softmax. Провести обучение и проверить, что модель адекватно предсказывает принадлежность произвольных точек плоскости (не из набора для обучения). Визуализировать эти предсказания с помощью разноцветных маркеров на плоскости (цвет марекера = номер класса).

Задание 5: Искуственно увеличить число нейронов во входном слое в несколько раз, а также число эпох для обучения модели. Часть обучающей выборки выделите для кроссвалидации. Это проще всего сделать, подав функции model.fit параметр validation_split = 0.2 (тогда 20 процентов из обучающей выборки будет использоваться только для валидации). Провести обучение и проверить, что модель переобучилась. Визуализировать наступление переобучения с помощью кривых для функции потерь, а также с помощью произвольных точек и разноцветных маркеров на плоскости.