Практическое задание №5, вариант 3

Выполнил: Облизанов Александр, гр. 8382

Последние цифры в зачетке: 15. Цель регрессии - 2 признак.

$X \in N(0,10)$
$e \in N(0,0.3)$

Признак	1	2	3	4	5	6	7
Формула	X^2+X+e	|X|+e	sin(X-PI\4)+e	lg(|X|)+e	-X^3+e	-X/4+e	-X+e

Генерация данных

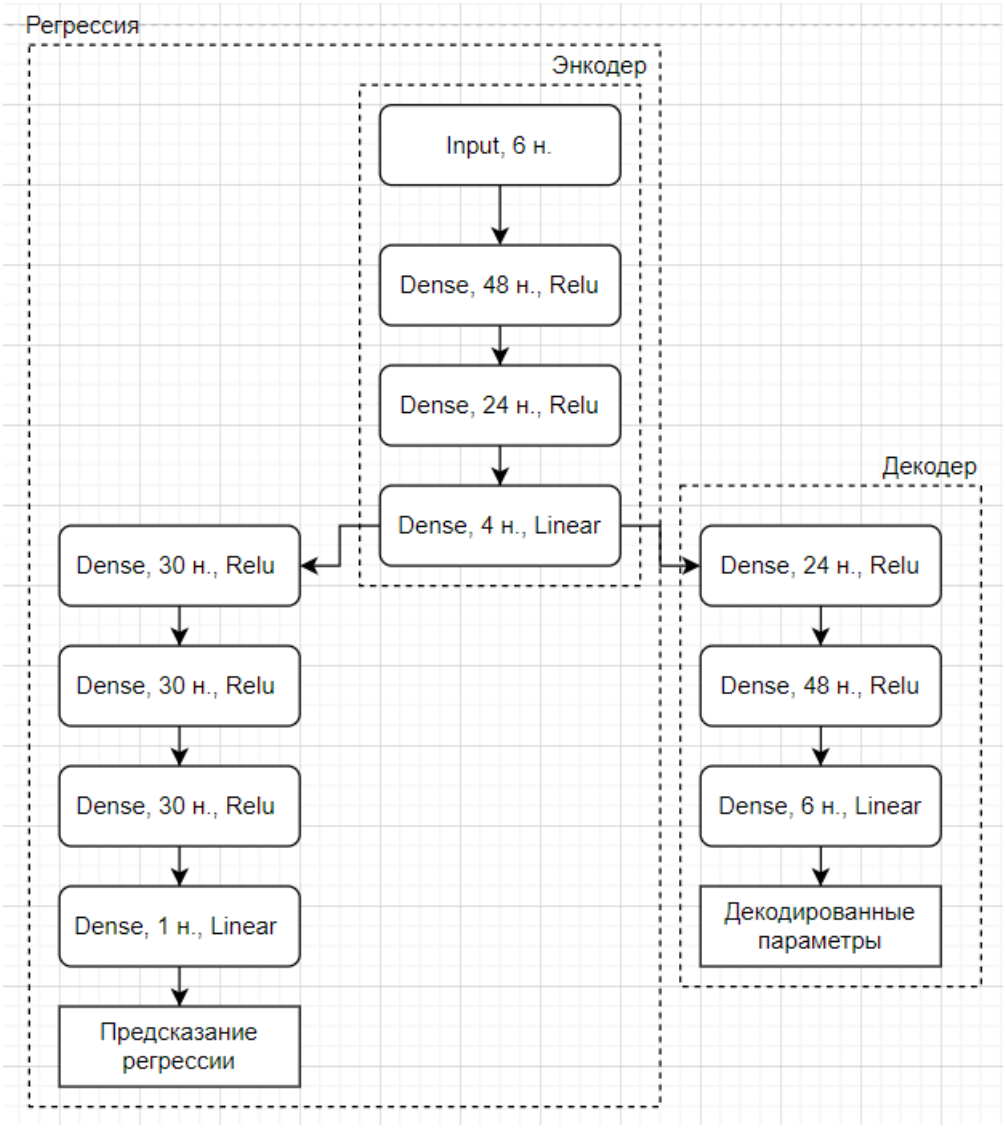
Данные генерируются функцией `gen_data`.

- Для обучения генерируется 500 строк данных
  - параметры в файле `train_data.csv`
  - цель регрессии в файле `train_labels.csv`
- Для проверки генерируется 50 строк данных
  - параметры в файле `test_data.csv`
  - цель регрессии в файле `test_labels.csv`

Подготовка данных, основная модель

Параметры для обучения и проверки были нормализованы: от каждого из параметров отнимается среднее значение параметра в данных для обучения и делится на СКО (также на основе данных для обучения)

Модель была построена в функциональном виде. Основная модель имеет следующую структуру:



Модель имеет два выхода: предсказание регрессии (одно число - 2 признак), декодированные параметры (шесть чисел, 1 и 3-7 признаки).

Параметры обучения модели:

- Оптимизатор: adam, функция потерь: mean squared error, распределение весов по выходам: 0.8 на декодирование, 0.2 на регрессию (было выяснено, что более высокие отклонения в результатах имеются именно в декодировании).
- Число эпох: 70, batch_size: 20

Результат обучения модели:

- Потери регрессии на данных для проверки: 0.0085
- Потери автоэнкодера на данных для проверки: 0.0036

Разделение модели

Энкодер и регрессия выделяются из основной сети просто, что можно увидеть на рисунке выше. Входной слой остается тем же, последними Dense слоями указываются "нижние" слои в соответствующем пунктирном прямоугольнике.

Для выделения декодера был создан входной слой с 4 нейронами (так как энкодер дает на выходе 4 значения), выход декодера соответствует "нижнему" слою на рисунке для декодера. Модель была построена с помощью функции `get_layer` из основной модели с данным входным слоем.

```
# creating standalone decoder model
decoder_input = Input(shape=(4,))
decoder_st = double_model.get_layer('dec1')(decoder_input)
decoder_st = double_model.get_layer('dec2')(decoder_st)
decoder_st = double_model.get_layer('decoded_output')(decoder_st)
decoder = Model(decoder_input, decoder_st)
```

Далее были выполнены запуски каждой из моделей (энкодер, декодер, регрессия) с помощью метода `predict` на данных для проверки, для декодера входными данными были результаты выполнения `predict` энкодера (то есть закодированные данные).

Далее описаны файлы, в которых отражены результаты тестирования:

- `encoder.csv` - результат `predict` энкодера
- `decoder.csv` - результат `predict` декодера (внимание, входные параметры были нормализованы, выход не совпадает с данными в `test_data.csv`)
- `decoder_denormalized.csv` - денормализованный результат `predict` декодера
- `regression.csv` - результат `predict` регрессии

Также сохраняется файл `test_data_nomalized.csv` с нормализованными параметрами, который можно сравнивать с файлом `decoder.csv`, так как модель работает именно с нормализованными данными.

Было произведено сохранение моделей:

- Модель декодера сохранена в файле `decoder_model.h5`
- Модель энкодера сохранена в файле `encoder_model.h5`
- Модель регрессии сохранена в файле `regression_model.h5`

Проверка

Сравним предсказание регрессии (`regression.csv`) с верными значениями 2-го признака (`test_labels.csv`) на нескольких строках:

regression.csv	**test_labels.csv**
9.424283981323242	9.410096435755761
15.469768524169922	15.527025847551531
1.1825428009033203	1.035381970372836
13.89572811126709	13.898632048730247

Как видно, значения близки.

Сравним выход декодера (`decoder.csv`) с нормализованными входными параметрами (`test_data_normalized.csv`) на нескольких строках:

```
decoder #1:
-0.020751506090164185,1.7507537603378296,0.6570203304290771,-0.24739238619804382
,-0.7421549558639526,-0.8354735970497131
test_data_normalized #1:
-0.07191501675639203,1.7574524275592338,0.8554070043889818,-0.18460861457241798,
-0.6668225808674605,-0.8123447401657565

decoder #2:
1.1095259189605713,1.1520601511001587,1.0621724128723145,-1.0898332595825195,-1.
4046605825424194,-1.4998695850372314
test_data_normalized #2:
1.1668251398680443,1.0755323278384932,0.9753659522396745,-1.0548515470890256,-1.
4848273505678315,-1.5115837368621245
```

Как видно, отклонения чуть более существенные (так как и сами значения меньше), но тем не менее значения близки.