Sequential model: руководство

Модель Sequential представляет собой линейный стек слоев.

Вы может создать модель Sequential, передав список слоев конструктору модели:

```
from keras.models import Sequential
from keras.layers import Dense, Activation
model = Sequential([Dense(32, input_shape=(784,)),
Activation('relu'),
Dense(10),
Activation('softmax'),])
```

Вы также можете добавить слои с помощью метода .add():

```
model = Sequential()
model.add(Dense(32, input_dim=784))
model.add(Activation('relu'))
```

Указание размерности входных данных

Ваша модель должна знать, какую размерность данных ожидать на входе. В связи с этим, первый слой модели Sequential (и только первый, поскольку последующие слои производят автоматический расчет размерности) должен получать информацию о размерности входных данных. Есть несколько способов сделать это:

- Передать аргумент input_shape первому слою (кортеж целых чисел или значений None, указывающих, что ожидается любое положительное целое число). Размер пакета (batch_size) в input_shape не включен.
- Некоторые 2D-слои (такие как Dense) поддерживают спецификацию размерности входных данных через аргумент input_dim. А некоторые 3D-слои поддерживают аргументы input_dim и input_length.
- Если вам когда-нибудь понадобится указать фиксированный размер пакета данных (batch_size) (это может быть полезно для рекуррентных

сетей с сохранением данных), вы можете передать аргумент batch_size слою. Если вы передадите оба параметра batch_size=32 и input_shape=(6, 8), то модель будет ожидать, что каждый пакет входных данных будет иметь размерность (32, 6 , 8).

Таким образом, следующие примеры эквивалентны:

```
model = Sequential()
model.add(Dense(32, input_shape=(784,)))
```

```
model = Sequential()
model.add(Dense(32, input_dim=784))
```

Компиляция

Перед обучением модели необходимо настроить сам процесс. Это выполняется с помощью метода compile(). Он получает три аргумента:

- Оптимизатор. Это может быть строковый идентификатор существующего оптимизатора (например, rmsprop или adagrad) или экземпляр класса Optimizer. Смотрите оптимизаторы.
- Функция ошибки. Это значение, которое модель пытается минимизировать. Это может быть строковый идентификатор существующей функции ошибки (например, categoryorc_crossentropy или mse) или собственная функция. Смотрите потери.
- Список метрик. Для любой задачи классификации вы можете установить это значение в metrics=[‘accuracy’]. Метрика может быть строковым идентификатором существующей метрики или пользовательской метрической функцией. Смотрите метрики.

```
# Задача многоклассовой классификации
model.compile(optimizer='rmsprop',
 loss='categorical_crossentropy', metrics=['accuracy'])

# Задача бинарной классификации
model.compile(optimizer='rmsprop',
 loss='binary_crossentropy', metrics=['accuracy'])
```

```
# Среднеквадратичная ошибка регрессии
model.compile(optimizer='rmsprop', loss='mse')

# Пользовательская метрика
import keras.backend as K
def mean_pred(y_true, y_pred):
return K.mean(y_pred)
model.compile(optimizer='rmsprop',
loss='binary_crossentropy', metrics=['accuracy', mean_pred])
```

Обучение

Модели Keras обучаются на Numpy-массивах, содержащих набор исходных данных и метки. Для обучения обычно используется функция fit(). Документация по этой функции здесь.

 # Модель с одномерными входными данными и бинарной классификацией

model = Sequential()

model.add(Dense(32, activation='relu', input_dim=100))

model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',

loss='binary_crossentropy',

metrics=['accuracy'])

 # Генерируем случайные данные

import numpy **as** np

data = np.random.random((1000, 100))

labels = np.random.randint(2, size=(1000, 1))

 # Обучаем модель, перебирая данные в пакетах по 32 примера

model.fit(data, labels, epochs=10, batch_size=32)

```
# Модель с одномерными входными данными и 10 классами

model = Sequential()

model.add(Dense(32, activation='relu', input_dim=100))

model.add(Dense(10, activation='softmax'))

model.compile(optimizer='rmsprop',

        loss='categorical_crossentropy',

        metrics=['accuracy'])

# Генерируем случайные данные

import numpy as np

data = np.random.random((1000, 100))

labels = np.random.randint(10, size=(1000, 1))

# Преобразуем метки в OHE (one-hot encoding)

one_hot_labels = keras.utils.to_categorical(labels, num_classes=10)

# Обучаем модель, перебирая данные в пакетах по 32 примера

model.fit(data, one_hot_labels, epochs=10, batch_size=32)
```

Примеры

Вот несколько примеров, с которых можно начать!

В папке примеров вы также найдете варианты решения задач с реальными наборами данных:

- Классификация небольших изображений из набора CIFAR10: сверточная нейронная сеть (CNN) с аугментацией данных в реальном времени.
- Классификация отзывов по фильмам IMDB: LSTM по последовательностям слов
- Тематическая классификация новостной ленты: многослойный персептрон (MLP)

- Классификация рукописных цифр набора MNIST: MLP & CNN
- Генерация текста на уровне символов с помощью LSTM

и другое…

Многослойный персептрон (MLP) для мультиклассовой классификаци (softmax):

```
import keras

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation

from keras.optimizers import SGD

# Генерируем случайные данные

import numpy as np

x_train = np.random.random((1000, 20))

y_train = keras.utils.to_categorical(
np.random.randint(10, size=(1000, 1)), num_classes=10)

x_test = np.random.random((100, 20))

y_test = keras.utils.to_categorical(
np.random.randint(10, size=(100, 1)), num_classes=10)

model = Sequential()

# Dense(64) — это полносвязный слой с 64 скрытыми нейронами.

# в первом слое вы должны указать размерность входных данных:

# здесь, это векторы длинной 20.

model.add(Dense(64, activation='relu', input_dim=20))

model.add(Dropout(0.5))
```

```
model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss='categorical_crossentropy',

        optimizer=sgd,

        metrics=['accuracy'])

model.fit(x_train, y_train,

      epochs=20,

      batch_size=128)

score = model.evaluate(x_test, y_test, batch_size=128)
```

MLP для бинарной классификации:

```
import numpy as np

from keras.models import Sequential

from keras.layers import Dense, Dropout

# Генерируем случайные данные

x_train = np.random.random((1000, 20))

y_train = np.random.randint(2, size=(1000, 1))

x_test = np.random.random((100, 20))

y_test = np.random.randint(2, size=(100, 1))

model = Sequential()

model.add(Dense(64, input_dim=20, activation='relu'))
```

```
model.add(Dropout(0.5))

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',

        optimizer='rmsprop',

        metrics=['accuracy'])

model.fit(x_train, y_train,

      epochs=20,

      batch_size=128)

score = model.evaluate(x_test, y_test, batch_size=128)
```

VGG-подобная сверточная сеть:

```
import numpy as np

import keras

from keras.models import Sequential

from keras.layers import Dense, Dropout, Flatten

from keras.layers import Conv2D, MaxPooling2D

from keras.optimizers import SGD

# Генерируем случайные данные

x_train = np.random.random((100, 100, 100, 3))

y_train = keras.utils.to_categorical(
np.random.randint(10, size=(100, 1)), num_classes=10)
```

```
x_test = np.random.random((20, 100, 100, 3))

y_test = keras.utils.to_categorical(
np.random.randint(10, size=(20, 1)), num_classes=10)

model = Sequential()

# Вход: изображение 100×100 с 3 каналами -> (100, 100, 3) тензор.

# применим здесь сверточный слой с 32 нейронами и ядром свертки (3, 3)

model.add(Conv2D(32, (3, 3), activation='relu',
input_shape=(100, 100, 3)))

model.add(Conv2D(32, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(256, activation='relu'))

model.add(Dropout(0.5))

model.add(Dense(10, activation='softmax'))

sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)

model.compile(loss='categorical_crossentropy', optimizer=sgd)

model.fit(x_train, y_train, batch_size=32, epochs=10)

score = model.evaluate(x_test, y_test, batch_size=32)
```

Классификация последовательностей с помощью LSTM:

```
from keras.models import Sequential

from keras.layers import Dense, Dropout

from keras.layers import Embedding

from keras.layers import LSTM

max_features = 1024

model = Sequential()

model.add(Embedding(max_features, output_dim=256))

model.add(LSTM(128))

model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',

        optimizer='rmsprop',

        metrics=['accuracy'])

model.fit(x_train, y_train, batch_size=16, epochs=10)

score = model.evaluate(x_test, y_test, batch_size=16)
```

Классификация последовательностей с помощью одномерной свертки:

```
from keras.models import Sequential

from keras.layers import Dense, Dropout

from keras.layers import Embedding
```

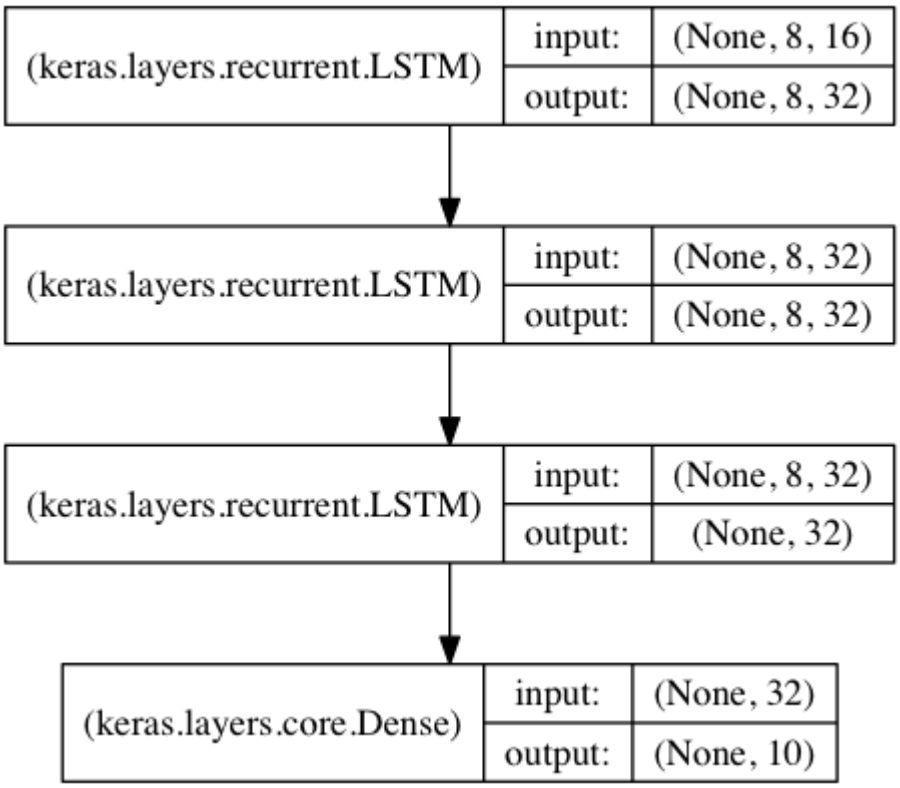
```
from keras.layers import Conv1D, GlobalAveragePooling1D, MaxPooling1D

seq_length = 64

model = Sequential()

model.add(Conv1D(64, 3, activation='relu',
input_shape=(seq_length, 100)))

model.add(Conv1D(64, 3, activation='relu'))

model.add(MaxPooling1D(3))

model.add(Conv1D(128, 3, activation='relu'))

model.add(Conv1D(128, 3, activation='relu'))

model.add(GlobalAveragePooling1D())

model.add(Dropout(0.5))

model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',

        optimizer='rmsprop',

        metrics=['accuracy'])

model.fit(x_train, y_train, batch_size=16, epochs=10)

score = model.evaluate(x_test, y_test, batch_size=16)
```

Классификация последовательностей с помощью LSTM с памятью:

В этой модели мы накладываем 3 слоя LSTM друг на друга, делая модель способной изучать временные представления более высокого уровня.

Первые два слоя возвращают свои полные выходные последовательности, но последний слой возвращает только последний шаг своей выходной

последовательности. Таким образом отбрасывается временное измерение (то есть входная последовательность преобразуется в один вектор).




```
from keras.models import Sequential

from keras.layers import LSTM, Dense

import numpy as np

data_dim = 16

timesteps = 8

num_classes = 10

# ожидаемая размерность входных данных:
# (batch_size, timesteps, data_dim)

model = Sequential()

model.add(LSTM(32, return_sequences=True,

        input_shape=(timesteps, data_dim)))
# возвращает последовательность векторов длинной 32

model.add(LSTM(32, return_sequences=True))
```

```
# возвращает последовательность векторов длинной 32

model.add(LSTM(32))  # возвращает одиночный векторов длинной 32

model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',

        optimizer='rmsprop',

        metrics=['accuracy'])

# Генерируем случайные данные

x_train = np.random.random((1000, timesteps, data_dim))

y_train = np.random.random((1000, num_classes))

# Генерируем случайные проверочные данные

x_val = np.random.random((100, timesteps, data_dim))

y_val = np.random.random((100, num_classes))

model.fit(x_train, y_train,

      batch_size=64, epochs=5,

      validation_data=(x_val, y_val))
```

LSTM с передачей состояния

Рекуррентная модель с состоянием — это модель, для которой внутренней состояние, полученное после обработки очередного пакета данных, повторно используется в качестве начальных состояний для выборок следующей серии. Это позволяет обрабатывать более длинные последовательности.

```
from keras.models import Sequential

from keras.layers import LSTM, Dense

import numpy as np
```

```
data_dim = 16

timesteps = 8

num_classes = 10

batch_size = 32

# ожидаемая размерность входных данных:
# (batch_size, timesteps, data_dim)

# Обратите внимание, что мы должны указать полную размерность входных

# данных batch_input_shape, так как это сеть с состоянием

# i-тый пример в k-ом пакете является продолжением

# i-того примера в k-1-ом пакете

model = Sequential()

model.add(LSTM(32, return_sequences=True, stateful=True,

          batch_input_shape=(batch_size, timesteps, data_dim)))

model.add(LSTM(32, return_sequences=True, stateful=True))

model.add(LSTM(32, stateful=True))

model.add(Dense(10, activation='softmax'))

model.compile(loss='categorical_crossentropy',

          optimizer='rmsprop',

          metrics=['accuracy'])

# Генерируем случайные данные

x_train = np.random.random((batch_size * 10, timesteps, data_dim))

y_train = np.random.random((batch_size * 10, num_classes))
```

 # Генерируем случайные проверочные данные

x_val = np.random.random((batch_size * 3, timesteps, data_dim))

y_val = np.random.random((batch_size * 3, num_classes))

model.fit(x_train, y_train,

batch_size=batch_size, epochs=5, shuffle=**False**,

validation_data=(x_val, y_val))