Модели Keras

О моделях Keras

В Keras существует два основных типа моделей: модель Sequential и класс Model, используемый с функциональным API.

Эти модели имеют ряд общих методов и атрибутов:

- model.layers это список слоев, составляющих модель..
- model.inputs это список входных тензоров модели.
- model.outputs это список выходных тензоров модели.
- model.summary() распечатывает сводное представление Вашей модели. Для слоев с несколькими выходами, multiple отображается вместо каждой отдельной выходной формы из-за ограничений по размеру.
- Ссылка на utils.print_summary
- model.get_config()возвращает словарь, содержащий конфигурацию модели. Модель может быть переустановлена из конфигурации через:

config = model.get_config()

model = Model.from_config(config)*# или, для Sequential:*

model = Sequential.from_config(config)

- model.get_weights() возвращает список всех тензоров веса в модели, в виде Numpy массивов.
- model.set_weights(weights) устанавливает значения весов модели из списка Numpy массивов. Массивы в списке должны иметь ту же форму, что и массивы, возвращаемые функцией get_weights().
- model.to_json() возвращает представление модели в виде JSON строки. Обратите внимание, что представление не включает в себя веса, а только архитектуру. Вы можете переустановить ту же самую модель (с повторно инициализированными весами) из JSON строки через:

```
from keras.models import model_from_json

json_string = model.to_json()

model = model_from_json(json_string)
```

- model.to_yaml() возвращает представление модели в виде YAML-строки. Обратите внимание, что представление не включает в себя веса, а только архитектуру. Вы можете переустановить ту же самую модель (с повторно инициализированными весами) из строки YAML via:

```
from keras.models import model_from_yaml

yaml_string = model.to_yaml()

model = model_from_yaml(yaml_string)
```

- model.save_weights(filepath) сохраняет веса модели в виде файла HDF5.
- model.load_weights(filepath, by_name=False) загружает веса модели из файла HDF5 (созданного с помощью save_weights). По умолчанию ожидается, что архитектура останется неизменной. Для загрузки весов в другую архитектуру (с общими для некоторых слоев) используйте by_name=True для загрузки только тех слоев с одинаковым именем.

Примечание: Как установить HDF5 или h5py для сохранения моделей в Keras? Смотрите FAQ для получения инструкций по установке h5py.

Подклассификация моделей

В дополнение к этим двум типам моделей, вы можете создавать свои собственные полностью настраиваемые модели, разместив класс Model и реализовав свою собственную переадресацию в методе вызова (API класса Model было введено в Keras 2.2.0).

Приведем пример простой многослойной модели перцептрона, написанной как подкласс Модель:

```
import keras

class SimpleMLP(keras.Model):

  def __init__(self, use_bn=False, use_dp=False, num_classes=10):
```

```
    super(SimpleMLP, self).__init__(name='mlp')
    self.use_bn = use_bn
    self.use_dp = use_dp
    self.num_classes = num_classes
    self.dense1 = keras.layers.Dense(32, activation='relu')
    self.dense2 = keras.layers.Dense(num_classes, activation='softmax')
    if self.use_dp:
      self.dp = keras.layers.Dropout(0.5)
    if self.use_bn:
      self.bn = keras.layers.BatchNormalization(axis=-1)
  def call(self, inputs):
    x = self.dense1(inputs)
    if self.use_dp:
      x = self.dp(x)
    if self.use_bn:
      x = self.bn(x)
    return self.dense2(x)
model = SimpleMLP()
model.compile(...)
model.fit(...)
```

Слои определены в __init__(self, ...), а в call(self, inputs) указан прямой проход. В вызове можно указать пользовательские потери, вызвав self.add_loss(loss_tensor) (как в пользовательском слое).

В подклассовых моделях топология модели определяется как код Python (а не как статический график слоев). Это означает, что топология модели не может быть проверена или сериализована. В результате, следующие методы и атрибуты недоступны для подклассовых моделей:

- model.inputs and model.outputs.
- model.to_yaml() and model.to_json()
- model.get_config() and model.save().

Ключевой момент: используйте правильный API для задания. API подкласса «Model» может предоставить вам большую гибкость для реализации сложных моделей, но это стоит дорого (в дополнение к этим отсутствующим возможностям): он более многословен, сложен и имеет больше возможностей для пользовательских ошибок. Если возможно, предпочитайте использовать функциональный API, который более удобен в использовании.