Advanced Keras API

Prof. Kuan-Ting Lai

2021/10/28
Keras Learning Stages

- Progressive complexity
Going Beyond Sequential Model

**Multi-input Model**

- Price prediction
- Merging module
- Dense module
- RNN module
- Convnet module
- Metadata
- Text description
- Picture

**Multi-output Model**

- Genre
- Date
- Genre classifier
- Date regressor
- Text-processing module
- Novel text
Inception Module

Residual Connection

Functional API

```python
from keras import Input, layers

input_tensor = Input(shape=(32,))
dense = layers.Dense(32, activation='relu')
output_tensor = dense(input_tensor)
```

A layer is a function

A layer may be called on a tensor, and it returns a tensor.
Functional API vs. Sequential Model

• Create a Model object using only an input tensor and an output tensor

```python
input_tensor = Input(shape=(64,))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(x)
model = Model(input_tensor, output_tensor)
```
Functional API vs. Sequential Model

• Create a Model object using only an input tensor and an output tensor

```python
input_tensor = Input(shape=(64,))
x = layers.Dense(32, activation='relu')(input_tensor)
x = layers.Dense(32, activation='relu')(x)
output_tensor = layers.Dense(10, activation='softmax')(x)
model = Model(input_tensor, output_tensor)
```

```python
seq_model = Sequential()
seq_model.add(layers.Dense(32, activation='relu',
                           input_shape=(64,)))
seq_model.add(layers.Dense(32, activation='relu'))
seq_model.add(layers.Dense(10, activation='softmax'))
```
Question-answering Model

• Two inputs:
  1. A natural-language question
  2. Reference text snippet (such as a news article)

• One output: answer
  – One-word answer obtained via a SoftMax over some predefined vocabulary
text_vocabulary_size = 10000
text_input = Input(shape=(None,), dtype='int32', name='text')
embedded_text = layers.Embedding(64, text_vocabulary_size)(text_input)
encoded_text = layers.LSTM(32)(embedded_text)

question_vocabulary_size = 10000
question_input = Input(shape=(None,), dtype='int32', name='question')
embedded_question = layers.Embedding(32, question_vocabulary_size)(question_input)
encoded_question = layers.LSTM(16)(embedded_question)

answer_vocabulary_size = 500

### Concatenate ###
concatenated = layers.concatenate([encoded_text, encoded_question], axis=-1)
answer = layers.Dense(answer_vocabulary_size, activation='softmax')(concatenated)

model = Model([text_input, question_input], answer)
model.compile(optimizer='rmsprop', loss='categorical_crossentropy', metrics=['acc'])
Train Two-input Models

- Training data can be array or dictionary

```python
import numpy as np
text = np.random.randint(1, text_vocabulary_size, size=(num_samples, max_length))
question = np.random.randint(1, question_vocabulary_size, size=(num_samples, max_length))
answers = np.random.randint(0, 1, size=(num_samples, answer_vocabulary_size))

### Option 1 ###
model.fit([text, question], answers, epochs=10, batch_size=128)

### Option 2 ###
model.fit({'text': text, 'question': question}, answers, epochs=10, batch_size=128)
```
Multi-output Model

• Predict age, income, gender based on the contents of posts
Multi-output Model

vocabulary_size = 50000
num_income_groups = 10

posts_input = Input(shape=(None,), dtype='int32', name='posts')
embedded_posts = layers.Embedding(256, vocabulary_size)(posts_input)
x = layers.Conv1D(128, 5, activation='relu')(embedded_posts)
x = layers.MaxPooling1D(5)(x)
x = layers.Conv1D(256, 5, activation='relu')(x)
...
x = layers.Dense(128, activation='relu')(x)

age_prediction = layers.Dense(1, name='age')(x)
income_prediction = layers.Dense(num_income_groups, activation='softmax', name='income')(x)
gender_prediction = layers.Dense(1, activation='sigmoid', name='gender')(x)

model = Model(posts_input, [age_prediction, income_prediction, gender_prediction])
### Option 1 ###
```python
model.compile(optimizer='rmsprop',
              loss=['mse', 'categorical_crossentropy', 'binary_crossentropy'])
```

### Option 2 ###
```python
model.compile(optimizer='rmsprop',
              loss={'age': 'mse',
                    'income': 'categorical_crossentropy',
                    'gender': 'binary_crossentropy'})
```
model.compile(optimizer='rmsprop',
    loss=['mse', 'categorical_crossentropy', 'binary_crossentropy'],
    loss_weights=[0.25, 1., 10.])

model.compile(optimizer='rmsprop',
    loss={'age': 'mse',
          'income': 'categorical_crossentropy',
          'gender': 'binary_crossentropy'},
    loss_weights={'age': 0.25, 'income': 1., 'gender': 10.})
Directed Acyclic Graph of Layers

• Graph can’t have cycles!
The Purpose of 1x1 Convolutions

• Reduce the channel dimension
Implement Inception Module

```python
from keras import layers

branch_a = layers.Conv2D(128, 1, activation='relu', strides=2)(x)

branch_b = layers.Conv2D(128, 1, activation='relu')(x)
branch_b = layers.Conv2D(128, 3, activation='relu', strides=2)(branch_b)

branch_c = layers.AveragePooling2D(3, strides=2)(x)
branch_c = layers.Conv2D(128, 3, activation='relu')(branch_c)

branch_d = layers.Conv2D(128, 1, activation='relu')(x)
branch_d = layers.Conv2D(128, 3, activation='relu')(branch_d)
branch_d = layers.Conv2D(128, 3, activation='relu', strides=2)(branch_d)

output = layers.concatenate([branch_a, branch_b, branch_c, branch_d], axis=-1)
```
from keras import layers

x = ...
y = layers.Conv2D(128, 3,
    activation='relu',
    padding='same')(x)
y = layers.Conv2D(128, 3,
    activation='relu',
    padding='same')(y)
y = layers.Conv2D(128, 3,
    activation='relu',
    padding='same')(y)
y = layers.add([y, x])
Vanishing Gradients in Deep Learning

• A signal becomes smaller after propagated through multi-layers, and may be lost (vanished)

• Solutions:
  – LSTM: using carry track to propagate signal parallel to main track
  – Residual: simple jump connection
Share layers and models

• Example: dual-camera

```python
from keras import layers
from keras import applications
from keras import Input

xception_base = applications.Xception(weights=None, include_top=False)

left_input = Input(shape=(250, 250, 3))
right_input = Input(shape=(250, 250, 3))
# Extract features from left and right cameras
left_features = xception_base(left_input)
right_features = xception_base(right_input)

merged_features = layers.concatenate([left_features, right_features], axis=-1)
```
Inheriting the Model Class

- Train a model to rank customer support tickets by priority
- In the `call()` method, define the forward pass of the model

```python
class CustomerTicketModel(keras.Model):
    def __init__(self, num_departments):
        super().__init__()
        self.concat_layer = layers.Concatenate()
        self.mixing_layer = layers.Dense(64, activation="relu")
        self.priority_scorer = layers.Dense(1, activation="sigmoid")
        self.department_classifier = layers.Dense(
            num_departments, activation="softmax")

    def call(self, inputs):
        title = inputs["title"]
        text_body = inputs["text_body"]
        tags = inputs["tags"]

        features = self.concat_layer([title, text_body, tags])
        features = self.mixing_layer(features)
        priority = self.priority_scorer(features)
        department = self.department_classifier(features)
        return priority, department
```
Test Our Custom Model

```python
model = CustomerTicketModel(num_departments=4)

priority, department = model({"title": title_data, "text_body": text_body_data, "tags": tags_data})

model.compile(optimizer="rmsprop",
               loss=["mean_squared_error", "categorical_crossentropy"],
               metrics=[["mean_absolute_error"], ["accuracy"]])

model.fit({"title": title_data,
            "text_body": text_body_data,
            "tags": tags_data},
           [priority_data, department_data],
           epochs=1)

model.evaluate({"title": title_data,
                 "text_body": text_body_data,
                 "tags": tags_data},
                [priority_data, department_data])

priority_preds, department_preds = model.predict({"title": title_data,
                                                  "text_body": text_body_data,
                                                  "tags": tags_data})
```
Monitoring Model Training

• Model checkpoint saving
  – Saving the current weights of the model during training

• Early stopping

• Dynamically adjusting parameters
  – Adaptive learning rate during training

• Visualizing the model and data
Using Callbacks

- **EarlyStopping** - interrupts training when accuracy has stopped improving for more than one epoch
- **ModelCheckpoint** - Saves the current weights after every epoch

```python
from keras import callbacks

callbacks_list = [
    callbacks.EarlyStopping(monitor='acc', patience=1),
    callbacks.ModelCheckpoint(filepath='my_model.h5', monitor='val_loss',
                              save_best_only=True)
]

model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
model.fit(x, y, epochs=10, batch_size=32, callbacks=callbacks_list,
          validation_data=(x_val, y_val))
```
ReduceLROnPlateau Callback

• factor – the learning rate is multiplied by factor after pre-defined epochs
• patience – epochs before callback is triggered

callbacks_list = [
    callbacks.ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=10)
]
model.fit(x, y, epochs=10, batch_size=32,
    callbacks=callbacks_list,
    validation_data=(x_val, y_val))
Implement Your Own Callback Function

- Inherit keras.callbacks.Callback and implement any number of the following methods
  - on_epoch_begin
  - on_epoch_end
  - on_batch_begin
  - on_batch_end
  - on_train_begin
  - on_train_end
Exmaple: Creating Your Own Logger

```python
class ActivationLogger(keras.callbacks.Callback):
    def set_model(self, model):
        self.model = model
        layer_outputs = [layer.output for layer in model.layers]
        self.activations_model = keras.models.Model(model.input, layer_outputs)

    def on_epoch_end(self, epoch, logs=None):
        if self.validation_data is None:
            raise RuntimeError('Requires validation_data.')
        validation_sample = self.validation_data[0][0:1]
        activations = self.activations_model.predict(validation_sample)
        f = open('activations_at_epoch_' + str(epoch) + '.npz', 'w')
        np.savez(f, activations)
        f.close()
```
• Add **TensorBoard** callback function and assign log_dir

```python
callbacks = [
    keras/callbacks.TensorBoard(log_dir='my_log_dir', histogram_freq=1, embeddings_freq=1)
]

history = model.fit(x_train, y_train, epochs=20, batch_size=128, validation_split=0.2, callbacks=callbacks)
```

• Run command => $ tensorboard --logdir=my_log_dir  --host localhost
Example: Text Classification with TensorBoard (2-1)

```python
import keras
import numpy as np
from keras import layers
from keras.datasets import imdb
from keras.preprocessing import sequence

max_features = 2000
max_len = 500
(x_train, y_train), (x_test, y_test) = imdb.load_data(num_words=max_features)
x_train = sequence.pad_sequences(x_train, maxlen=max_len)
x_test = sequence.pad_sequences(x_test, maxlen=max_len)

model = keras.models.Sequential()
model.add(layers.Embedding(max_features, 128, input_length=max_len, name='embed'))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.MaxPooling1D(5))
model.add(layers.Conv1D(32, 7, activation='relu'))
model.add(layers.GlobalMaxPooling1D())
model.add(layers.Dense(1))
model.summary()
```
Example: Text Classification with TensorBoard (2-2)

```python
model.compile(optimizer='rmsprop',
               loss='binary_crossentropy',
               metrics=['acc'])

callbacks = [
    keras.callbacks.TensorBoard(
        log_dir='my_log_dir',
        histogram_freq=1,
        embeddings_freq=1,
        embeddings_data = np.arange(0, max_len).reshape((1, max_len)),
    )
]

history = model.fit(x_train, y_train,
                     epochs=20,
                     batch_size=128,
                     validation_split=0.2,
                     callbacks=callbacks)
```
TensorBoard: Accuracy and Loss
TensorBoard: Activation Histograms
TensorBoard: Word-embedding Visualization
TensorBoard: Network Graph Visualization
from keras.utils import plot_model

plot_model(model, show_shapes=True, to_file='model.png')
Batch Normalization


- Normalizing data after every transformation
- Enhance back propagation
- Some deep networks can only be trained with batch normalization

```python
conv_model.add(layers.Conv2D(32, 3, activation='relu'))
conv_model.add(layers.BatchNormalization())

dense_model.add(layers.Dense(32, activation='relu'))
dense_model.add(layers.BatchNormalization())
```
Batch Renormalization


Depthwise Separable Convolution

• Separating the learning of spatial features and channel-wise features
• Less parameters, slightly better accuracy
• Francois Chollet, “Xception: Deep Learning with Depthwise Separable Convolutions,” CVPR, 2018
Xception - Separable Convolution (2017)

• Assume that cross-channel correlations and spatial correlations can be mapped completely separately

https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728
Normal Convolution of 1 Kernel

- Normal Convolution of 5*5*3 kernel

https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728
Normal Convolution of 256d Output Channels

- Require $256 \times (5\times5\times3+1) = 19,456$ parameters!

https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728
Separable Convolution

• Two steps:
  1. Depthwise: (5*5*1)*3
  2. Pointwise: (1*1*3+1)*256
Normal Convolution vs. Depthwise Separable

• Normal filter size:
  \[(5*5*3 + 1)*256\]  
  = 19,456

  *Bias=1

• Depthwise + Pointwise filter size:
  \[(5*5*1)*3 + (1*1*3+1)*256\]  
  = 1099

  *Bias=1
Test Separable Convolution in Colab

- **SeparableConv2D()**
Hyperparameter Optimization

- Use random search, genetic algorithm, Bayesian optimization to find best parameters for your model
- Hyperopt (https://github.com/hyperopt/hyperopt)
- Hyperas (https://github.com/maxpumperla/hyperas)
  - Varying dropout probabilities, sampling from a uniform distribution
  - Different layer output sizes
  - Different optimization algorithms to use
  - Varying choices of activation functions
  - Conditionally adding layers depending on a choice
  - Swapping whole sets of layers
Model Ensembling

• Combine the outputs of multiple models (a.k.a late fusion)
  – Random forest
  – Gradient-boosted trees
  – Wide and deep model
Key Takeaways

• Use Keras API to go beyond `Sequential()`.
• Keras callbacks provide a simple way to monitor model training.
• You can control what `fit()` does by overriding the `train_step()` method.
• Beyond `fit()`, researchers can write their own training loops entirely from scratch for brand-new training algorithms.
• Separable convolution can reduce model parameters with same accuracy.
References

• Francois Chollet, “Deep Learning with Python,” Chapter 7
• Francois Chollet, “Deep Learning with Python, 2nd Edition” Chapter 7
• https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728