Introduction to Keras

Francois Chollet
March 9th, 2018
Outline (45 min + questions)

- What’s Keras?
  - What’s special about it?
  - TensorFlow integration
- How to use Keras
  - 3 API styles
  - An image captioning example
- Distributed, multi-GPU, and TPU training
- Eager execution (a.k.a define-by-run, a.k.a. dynamic graphs)
What’s Keras?
Keras: an API for specifying & training differentiable programs

Keras API

TensorFlow / CNTK / MXNet / Theano / ...

GPU  CPU  TPU
Keras is the official high-level API of TensorFlow

- tensorflow.keras (tf.keras) module
- Part of core TensorFlow since v1.4
- Full Keras API
- Better optimized for TF
- Better integration with TF-specific features
  - Estimator API
  - Eager execution
  - etc.
Who makes Keras? Contributors and backers

633 contributors

Google

Microsoft

NVIDIA

aws
What’s special about Keras?

- A focus on user experience.
- Large adoption in the industry and research community.
- Multi-backend, multi-platform.
- Easy productization of models.
Hacker News jobs board mentions - out of 964 job postings
Research traction

arXiv mentions as of 2018/03/07 (past 3 months)

arXiv mentions as of 2018/03/07 (past 1 month)
Keras is an API designed for human beings, not machines. Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

This makes Keras easy to learn and easy to use. As a Keras user, you are more productive, allowing you to try more ideas than your competition, faster -- which in turn helps you win machine learning competitions.

This ease of use does not come at the cost of reduced flexibility: because Keras integrates with lower-level deep learning languages (in particular TensorFlow), it enables you to implement anything you could have built in the base language. In particular, as tf.keras, the Keras API integrates seamlessly with your TensorFlow workflows.
Keras is multi-backend, multi-platform

- Develop in Python, R
  - On Unix, Windows, OSX
- Run the same code with...
  - TensorFlow
  - CNTK
  - Theano
  - MXNet
  - PlaidML
  - ??
- CPU, NVIDIA GPU, AMD GPU, TPU...
Largest array of options for productizing models

- TF-Serving
- In-browser, with GPU acceleration (WebKeras, Keras.js, WebDNN...)
- Android (TF, TF Lite), iPhone (native CoreML support)
- Raspberry Pi
- JVM

Go build cool AR apps with Keras + TF + CoreML + ARKit
How to use Keras: An introduction
Three API styles

- The Sequential Model
  - Dead simple
  - Only for single-input, single-output, sequential layer stacks
  - Good for 70+% of use cases

- The functional API
  - Like playing with Lego bricks
  - Multi-input, multi-output, arbitrary static graph topologies
  - Good for 95% of use cases

- Model subclassing
  - Maximum flexibility
  - Larger potential error surface
```python
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```
The functional API

```python
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```
import keras
from keras import layers

class MyModel(keras.Model):
    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
Remember: use the right tool (API) for the job!
Example: building a video captioning model
Toy video-QA problem

- “What is the man doing?”
  - packing

- “What color is his shirt?”
  - blue
from frames to a vector

video vector

LSTM

CNN

frame

video
answer word

Classifier

Concat

LSTM

CNN

CNN

CNN

frame frame frame

video

question

Embed

LSTM
Turning frames into a vector, with pre-trained representations

```python
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                 include_top=False,
                 pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```
Turning frames into a vector, with pre-trained representations

```python
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
    include_top=False,
    pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```
Turning frames into a vector, with pre-trained representations

```python
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                  include_top=False,
                  pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```
Turning frames into a vector, with pre-trained representations

```python
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                   include_top=False,
                   pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```
Turning frames into a vector, with pre-trained representations

```python
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                  include_top=False,
                  pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```
Turning a sequence of words into a vector

```python
question = keras.Input(shape=(None,), dtype='int32', name='question')
embedded_words = layers.Embedding(input_voc_size, 256)(question)
question_vector = layers.LSTM(128)(embedded_words)
```
Predicting an answer word

```python
x = layers.concatenate([[video_vector, question_vector]])
x = layers.Dense(128, activation=tf.nn.relu)(x)
predictions = layers.Dense(output_voc_size,
    activation='softmax',
    name='predictions')(x)
```
Setting up the training configuration

```python
model = keras.models.Model([video, question], predictions)
model.compile(optimizer=tf.AdamOptimizer(),
               loss=keras.losses.categorical_crossentropy)
model.fit_generator(data_generator,
                     steps_per_epoch=1000,
                     epochs=100)
```
Distributed, multi-GPU, & TPU training
Distributed

- Uber’s Horovod
- Estimator API (TF built-in option)
- Dist-Keras (Spark)
  - Also Elephas (Spark)
import tensorflow as tf
from keras.applications import Xception
from keras.utils import multi_gpu_model

# Instantiate the base model
# (here, we do it on CPU, which is optional).
with tf.device('/cpu:0'):
    model = Xception(weights=None,
                      input_shape=(height, width, 3),
                      classes=num_classes)

# Replicates the model on 8 GPUs.
# This assumes that your machine has 8 available GPUs.
parallel_model = multi_gpu_model(model, gpus=8)
parallel_model.compile(loss='categorical_crossentropy',
                       optimizer='rmsprop')

# This `fit` call will be distributed on 8 GPUs.
# Since the batch size is 256, each GPU will process 32 samples.
parallel_model.fit(x, y, epochs=20, batch_size=256)
TPU support

Training + inference

Via Estimator API
Eager execution
Understanding deferred (symbolic) vs. eager (imperative)

**Deferred**: you use Python to build a computation graph that gets executed later

**Eager**: the Python runtime is the execution runtime (like Numpy)

In short:

- Symbolic tensors **don’t have a value** in your Python code (yet)
- Eager tensors **have a value** in your Python code
- With eager execution, you can use value-dependent dynamic topologies (tree-RNNs)
The Keras functional API and Sequential API work with eager execution

```python
import keras
from keras import layers
import tensorflow.contrib.eager as tfe

tfe.enable_eager_execution()

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)

preds = model(x)  # Works on *value arrays*!
model.fit(x, y, epochs=10, batch_size=32)  # The Model API works too.
```
Eager execution allows you to write imperative custom layers

class DynamicLayer(tf.keras.Layer):

...

def call(self, x):
    print(x)  # debug!
    if tf.reduce_sum(x) == 0:
        # dynamic if!
        return x

    for t in x.shape[1]:
        # dynamically iterate over time axis!
        ...

    return outputs
Maximum flexibility: imperative Model subclassing

class MyRNN(keras.Model):

    def __init__(self, units=10):
        super(MyRNN, self).__init__(name='my_rnn')
        self.units = units
        self.dense_1 = keras.layers.Dense(units, activation='relu')
        self.dense_2 = keras.layers.Dense(units, activation='relu')

    def call(self, inputs):
        outputs = []
        state = tf.zeros(shape=(inputs.shape[0], self.units))
        for t in range(inputs.shape[1]):
            x = inputs[:, t, :]
            h = self.dense_1(x)
            y = h + self.dense_2(state)
            state = y
            outputs.append(y)
        return tf.concat(outputs, axis=1)
That’s it.
Thank you!