Keras: Deep Learning library for Theano and TensorFlow

BIL 722: Advanced Topics in Computer Vision

Mehmet Günel

Why this name, Keras?

- Keras (κέρας) means horn in Greek
- It is a reference to a literary image from ancient Greek and Latin literature
- Two divided dream spirits;
  - Ivory, those who deceive men with false visions
  - Horn, those who announce a future that will come to pass

Overview of KERAS

- Minimalist, highly modular neural networks library
- Written in Python
- Capable of running on top of either TensorFlow or Theano
- Developed with a focus on enabling fast experimentation

Guiding principles

- Modularity
  - A model is understood as a sequence or a graph of standalone, fully-configurable modules that can be plugged together with as little restrictions as possible. In particular, neural layers, cost functions, optimizers, initialization schemes, activation functions, regularization schemes are all standalone modules that you can combine to create new models.

Guiding principles

- Minimalism
  - Each module should be kept short and simple. Every piece of code should be transparent upon first reading. No black magic: it hurts iteration speed and ability to innovate.

Guiding principles

- Easy extensibility
  - New modules are dead simple to add (as new classes and functions), and existing modules provide ample examples. To be able to easily create new modules allows for total expressiveness, making Keras suitable for advanced research.

Guiding principles

- Work with Python
  - No separate models configuration files in a declarative format. Models are described in Python code, which is compact, easier to debug, and allows for ease of extensibility.

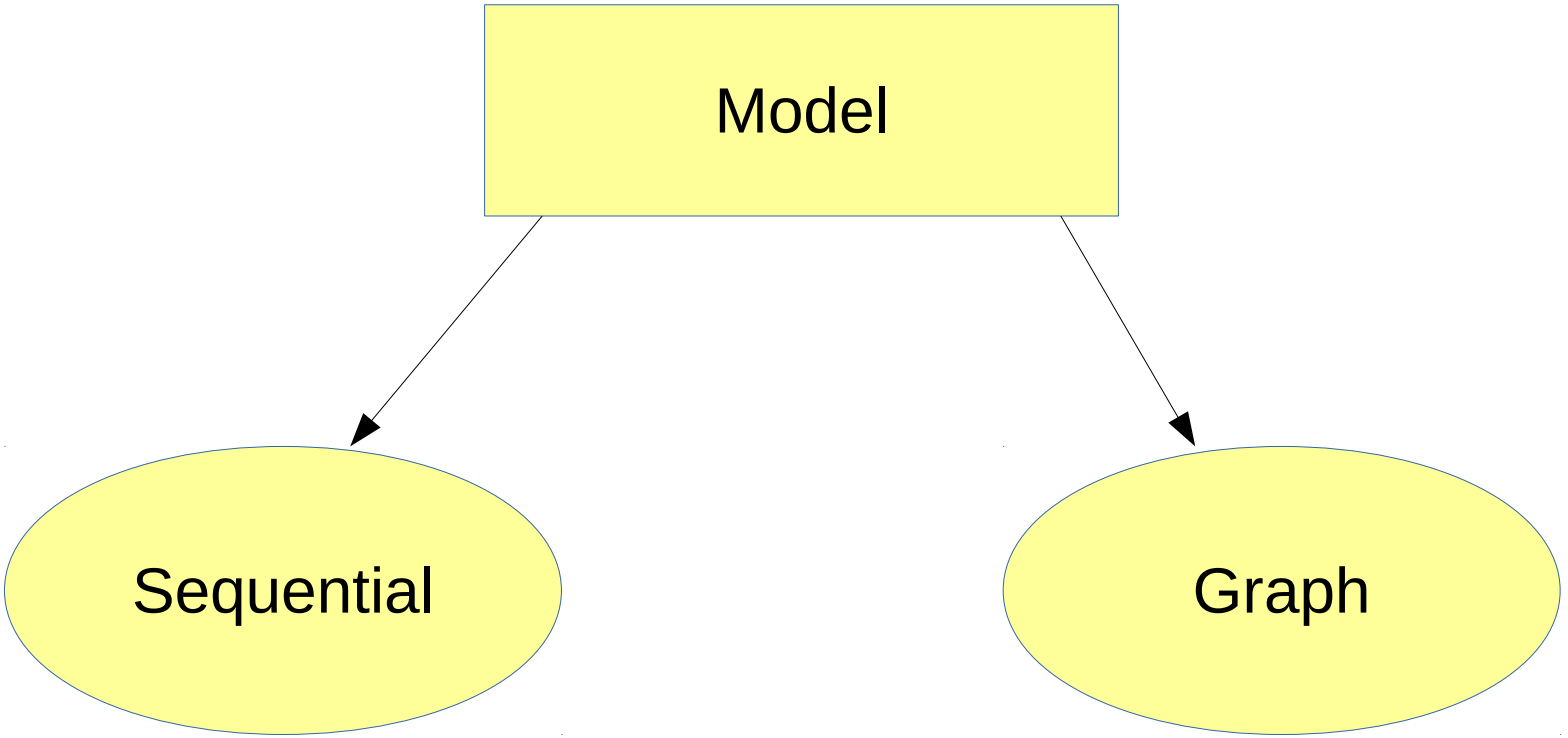
When to use KERAS?

- Easy and fast prototyping (through total modularity, minimalism, and extensibility).
- Supports both convolutional networks and recurrent networks and combinations of the two.
- Supports arbitrary connectivity schemes (including multi-input and multi-output training).
- Runs seamlessly on CPU and GPU.

Code Examples

- The core data structure of Keras is a **model**

  Model → a way to organize layers



Code Examples

- Sequential model (a linear pile of layers):

```
from keras.models import Sequential
model = Sequential()
```

```
from keras.layers.core import Dense, Activation

model.add(Dense(output_dim=64, input_dim=100,
init="glorot_uniform"))
model.add(Activation("relu"))
model.add(Dense(output_dim=10, init="glorot_uniform"))
model.add(Activation("softmax"))
```

Code Examples

- Compile model

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

```
from keras.optimizers import SGD

model.compile(loss='categorical_crossentropy',
optimizer=SGD(lr=0.01, momentum=0.9, nesterov=True))
```

Code Examples

- Training

```
model.fit(X_train, Y_train, nb_epoch=5, batch_size=32)
```

```
model.train_on_batch(X_batch, Y_batch)
```

Code Examples

- Evaluation

```
objective_score = model.evaluate(X_test, Y_test,
batch_size=32)
```

- Prediction

```
classes = model.predict_classes(X_test, batch_size=32)
proba = model.predict_proba(X_test, batch_size=32)
```

Multilayer Perceptron (MLP) for multi-class softmax classification

```
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation
from keras.optimizers import SGD

model = Sequential()

model.add(Dense(64, input_dim=20, init='uniform'))
model.add(Activation('tanh'))
model.add(Dropout(0.5))
model.add(Dense(64, init='uniform'))
model.add(Activation('tanh'))
model.add(Dropout(0.5))
model.add(Dense(10, init='uniform'))
model.add(Activation('softmax'))

sgd = SGD(lr=0.1, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy',
              optimizer=sgd)

model.fit(X_train, y_train, nb_epoch=20, batch_size=16,
          show_accuracy=True)
score = model.evaluate(X_test, y_test, batch_size=16)
```

Code Examples, Graph Model

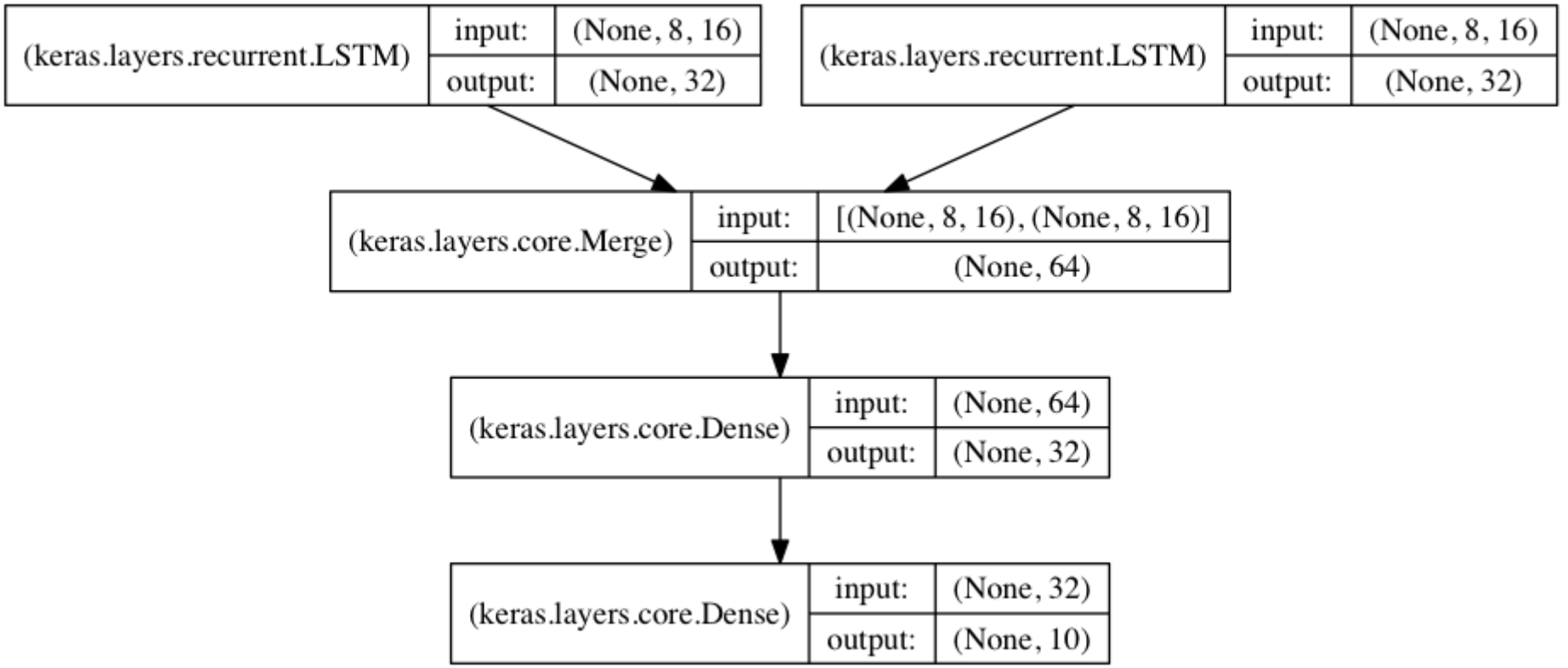
```
# graph model with one input and two outputs

graph = Graph()
graph.add_input(name='input', input_shape=(32,))
graph.add_node(Dense(16), name='dense1', input='input')
graph.add_node(Dense(4), name='dense2', input='input')
graph.add_node(Dense(4), name='dense3', input='dense1')
graph.add_output(name='output1', input='dense2')
graph.add_output(name='output2', input='dense3')

graph.compile(optimizer='rmsprop', loss={'output1':'mse',
'output2':'mse'})

history = graph.fit({'input':X_train, 'output1':y_train,
'output2':y2_train}, nb_epoch=10)
```

Layer Visualization



How to install

- Dependencies:
  - numpy, scipy
  - pyyaml
  - HDF5 and h5py (optional, required if you use model saving/loading functions)
  - (Optional) cuDNN

How to install

- *TensorFlow backend:*
  - *See https://github.com/tensorflow/tensorflow#download-and-setup*

- *Theano backend: (do not use PyPI version)*
  - *sudo pip install git+git://github.com/Theano/Theano.git*
  - *(By default, Keras will use Theano as its tensor manipulation library)*

How to install

- Simple installation
  - *sudo python setup.py install*

    Or

  - *sudo pip install keras*

Conclusion

- Deep CNN
- LSTM
- RNN
- Neural Turing Machine
- word2vec embedder
- ...

References

- http://keras.io/
- https://github.com/Theano/Theano
- https://github.com/tensorflow/tensorflow