STAT430: Machine Learning for Financial Data

LarryHua.com/teaching

Spring 2019

Keras functional API

Introduction to Keras functional API

- Keras objects are modified in place, not necessary to be assigned back, eg:

```
network %>% compile(optimizer = "rmsprop",loss = "categorical_crossentropy",metrics = c("accuracy"))
```

- With Keras, we can treat any model as a layer: re-use the architecture and the weights
- With Keras functional API, we can create models that
  - can easily process sequences of inputs
    - `time_distributed()`
      - apply a layer to every temporal slice of an input
      - the axis right after the sample axis is the temporal axis
  - have multi-inputs and multi-outputs
    - `layer_concatenate()` combines multiple inputs
  - have shared layers
  - represent directed acyclic graphs
- Try R

Hierarchical RNN (HRNN)

- HRNN is an example of using Keras API to process sequences of inputs
- HRNNs can learn across multiple levels of temporal hierarchy
- Example of using HRNN for text analysis
  - the 1st recurrent layer of an HRNN encodes a sentence (e.g. of word vectors) into a sentence vector
  - the 2nd recurrent layer then encodes a sequence of such vectors (encoded by the first layer) into a document vector
  - the document vector is considered to preserve both the word-level and sentence-level structure of the context

Hierarchical RNN (HRNN) - Examples

- Example of using HRNN to classify MNIST digits
  - the first LSTM layer encodes every row of pixels of shape (28, 1) to a vector of shape (128)
  - the second LSTM layer then encodes these 28 vectors to an image vector representing the whole image
  - a final dense layer is added for prediction
- Try R
- Example of using HRNN to LOB analysis
  - the first LSTM layer encodes every time step of shape (20, 1) to a vector of shape (32)
  - the second LSTM layer then encodes these 100 (w) vectors to a vector to represent the whole spatial-temporal image
  - a final dense layer is added for prediction
- Try R

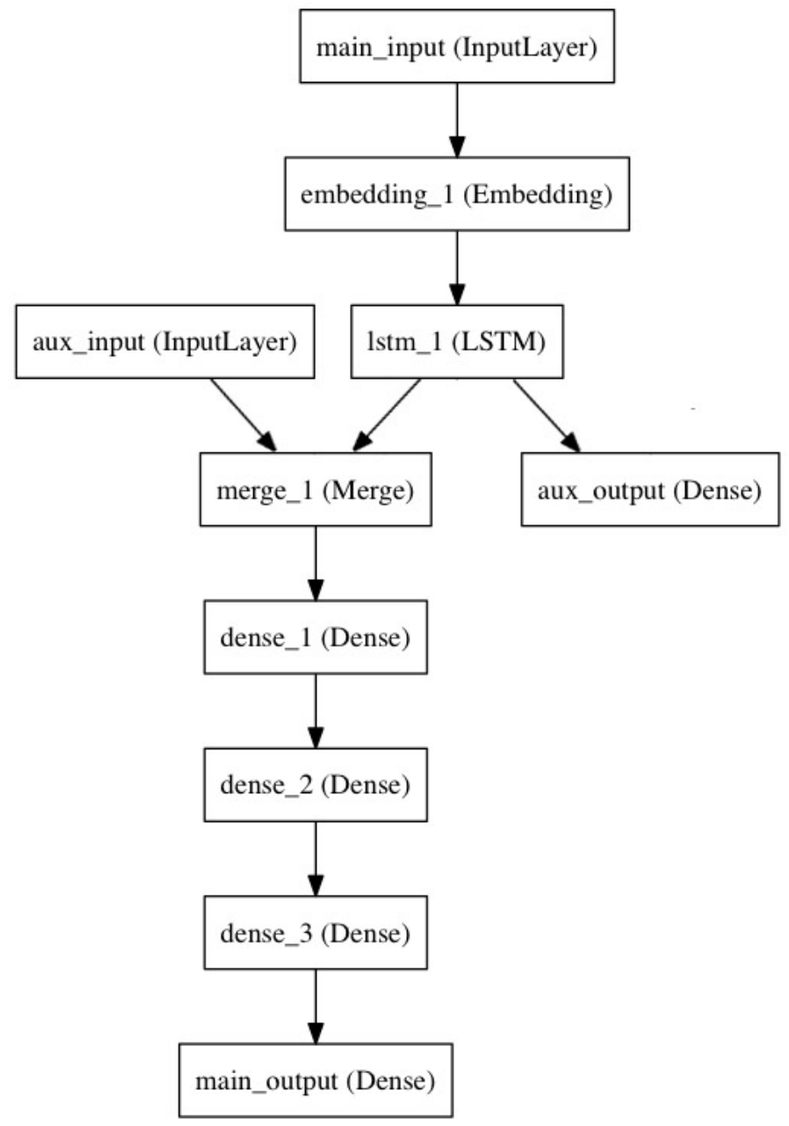
Multiple input / output models

- use `layer_concatenate()` to combine multiple inputs
  - eg, `layer_concatenate(c(lstm_out, auxiliary_input))`
- use `loss_weights` to assign weights for multiple loss functions of multiple outputs
  - eg, `model %>% compile(..., loss_weights = c(1.0, 0.2), ...)`
    - the order of weights is based on `outputs` specified in `keras_model()`
  - alternatively, `model %>% compile(..., loss_weights = list(main_output = 1.0, aux_output = 0.2), ...)`
    - `main_output` and `aux_output` are the names given for the corresponding layers
- use `loss` to assign loss functions of multiple outputs
  - eg, `model %>% compile(..., loss = list(main_output = 'binary_crossentropy', aux_output = 'binary_crossentropy'), ...)`

Multiple input / output models - Example

- Predict numbers of retweets and likes for a news headline
  - the main input is the headline itself, as a sequence of words
  - the auxiliary input is the time of day when the headline was posted
  - the model is supervised via two loss functions

Multiple input / output models - Example



- Try R

Shared layers

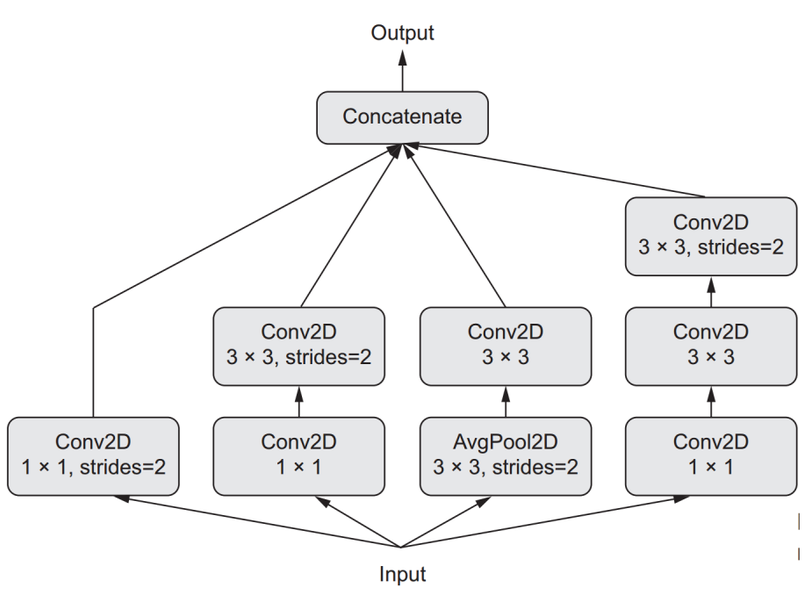
- instantiate a layer once, then call it on as many inputs as you want
- the weights of the shared layer are also reused
- Example: predict whether two tweets are from the same person or not
  - the mechanism that encodes tweets a and b should be the same
    - `encoded_a <- tweet_a %>% shared_lstm`
    - `encoded_b <- tweet_b %>% shared_lstm`
- Try R

Directed acyclic graphs of layers

- Keras allows arbitrary architectures that can be represented as directed acyclic graphs
- Two notable ones:
  - Inception modules
  - Residual connections

Inception modules

- a popular type of network architecture for convolutional neural networks
- consists of a stack of modules that themselves look like small independent parallel network branches
- the most basic form of an Inception module has 3 to 4 branches starting with a 1 × 1 convolution, followed by a 3 × 3 convolution, and ending with the concatenation of the resulting features
- **learn spatial features and channel-wise features separately, which is more efficient than learning them jointly**, why?
- after concatenation, the number of channels of the output is the sum of the numbers of channels of the branches



Inception modules - 1×1 convolution

- pointwise convolution, i.e., 1×1 convolution, contributes to factoring out channel-wise feature learning and space-wise feature learning
- running each single tile vector through a dense layer: mixing channels but not across space
  - revisit the toy example
- useful if each channel is highly auto-correlated across space, but different channels may not be highly correlated with each other
- Try R
- Back to Course Scheduler