Model_HRNN

YU Hong

2018/12/12

Model Hierarchical Rnn

HRNNs can learn across multiple levels of temporal hiearchy over a complex sequence. Usually, the first recurrent layer of an HRNN encodes a sentence (e.g. of word vectors) into a sentence vector. The second recurrent layer then encodes a sequence of such vectors (encoded by the first layer) into a document vector. This document vector is considered to preserve both the word-level and sentence-level structure of the context.

In the below MNIST example the first LSTM layer first encodes every column of pixels of shape (28, 1) to a column vector of shape (128,). The second LSTM layer encodes then these 28 column vectors of shape (28, 128) to a image vector representing the whole image. A final dense layer is added for prediction.

Data Preparation

```r
library(keras)
batch_size <- 128
num_classes <- 10
epochs <- 5
```

Input image dimensions

```r
img_rows <- 28
img_cols <- 28
```

The data, shuffled and split between train and test sets

```r
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y
```

Redefine dimension of train/test inputs

```r
x_train <- array_reshape(x_train, c(nrow(x_train), img_rows, img_cols, 1))
x_test <- array_reshape(x_test, c(nrow(x_test), img_rows, img_cols, 1))
input_shape <- c(img_rows, img_cols, 1)
```

Transform RGB values into [0,1] range

```r
x_train <- x_train / 255
x_test <- x_test / 255
cat('x_train_shape:', dim(x_train), '\n')
```

```
## x_train_shape: 60000 28 28 1
```

```r
cat(nrow(x_train), 'train samples\n')
```

```
## 60000 train samples
```

```r
cat(nrow(x_test), 'test samples\n')
```

```
## 10000 test samples
```

Convert class vectors to binary class matrices

```r
y_train <- to_categorical(y_train, num_classes)
y_test <- to_categorical(y_test, num_classes)
```

Embedding dimensions

```r
# Embedding dimensions.
row_hidden <- 128
col_hidden <- 128
```

Define layers of model

```r
# Model input (4D)
input <- layer_input(shape = input_shape)

# Encodes a row of pixels using TimeDistributed Wrapper
encoded_rows <- input %>% time_distributed(layer_lstm(units = row_hidden))

# Encodes columns of encoded rows
encoded_columns <- encoded_rows %>% layer_lstm(units = col_hidden)

# Model output
prediction <- encoded_columns %>%
  layer_dense(units = num_classes, activation = 'softmax')
```

Define Model

Compile model

```r
model <- keras_model(input, prediction)
model %>% compile(
  loss = loss_categorical_crossentropy,
  optimizer = optimizer_adadelta(),
  metrics = c('accuracy')
)
```

Train model

```
history<-model %>% fit(
  x_train, y_train,
  batch_size = batch_size,
  epochs = epochs,
  validation_split = 0.2
)
```

Result

```
scores_hrnn <- model %>% evaluate(
  x_test, y_test, verbose = 0
)
scores_hrnn
```

```
## $loss
## [1] 0.0712801
##
## $acc
## [1] 0.9784
```