Deep Learning Lecture - text processing and multi-input model

MA8701 General Statistical Methods

Thiago G. Martins, Department of Mathematical Sciences, NTNU

Spring 2019

Using Keras for word-level one-hot encoding

- Create a tokenizer and build a word index

```
samples <- c("The cat sat on the mat.", "The dog ate my homework.")
tokenizer <- text_tokenizer(num_words = 1000) %>%
  fit_text_tokenizer(samples)
```

- Turns strings into lists of integer indices

```
sequences <- texts_to_sequences(tokenizer, samples)
```

- You could also directly get the one-hot binary representations. Vectorization modes other than one-hot encoding are supported by this tokenizer: “binary”, “count”, “tfidf”, “freq”.

```
one_hot_results <- texts_to_matrix(tokenizer, samples, mode = "binary")
```

- We can recover the word index that was computed.

```
word_index <- tokenizer$word_index
cat("Found", length(word_index), "unique tokens.\n")
```

Loading pretrained word embeddings into the embedding layer

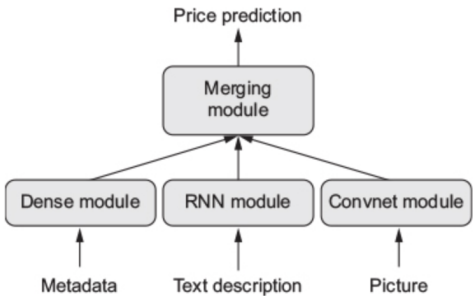
```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_words, output_dim = embedding_dim,
                  input_length = maxlen) %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 1, activation = "sigmoid")
summary(model)
```

- Load weights into embedding layer and freeze them

```
get_layer(model, index = 1) %>%
  set_weights(list(embedding_matrix)) %>%
  freeze_weights()
```

Keras functional API

- Example: multi-input models



- Comparison between sequential and functional API

```
# sequential API
seq_model <- keras_model_sequential() %>%
  layer_dense(units = 32, activation = "relu", input_shape = c(64)) %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")
```

- Note that the output_tensor depends explicitly on the input_tensor, otherwise it would have been impossible for keras to understand how they are connected.

```
# functional API
input_tensor <- layer_input(shape = c(64))
output_tensor <- input_tensor %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 32, activation = "relu") %>%
  layer_dense(units = 10, activation = "softmax")
model <- keras_model(input_tensor, output_tensor)
```

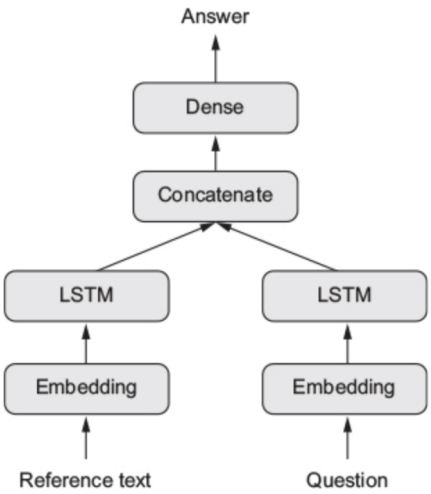
- Nothing changes wrt compiling and training the model

```
model %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy"
```

```
)
x_train <- array(runif(1000 * 64), dim = c(1000, 64))
y_train <- array(runif(1000 * 10), dim = c(1000, 10))
model %>% fit(x_train, y_train, epochs = 10, batch_size = 128)
model %>% evaluate(x_train, y_train)
```

Multi-input models

- A question-answering model example




```
# constants
text_vocabulary_size <- 10000
ques_vocabulary_size <- 10000
answer_vocabulary_size <- 500

# text input
text_input <- layer_input(shape = list(NULL),
                          dtype = "int32", name = "text")
encoded_text <- text_input %>%
  layer_embedding(input_dim = 64, output_dim = text_vocabulary_size) %>%
  layer_lstm(units = 32)

# question input
question_input <- layer_input(shape = list(NULL),
                              dtype = "int32", name = "question")
encoded_question <- question_input %>%
  layer_embedding(input_dim = 32, output_dim = ques_vocabulary_size) %>%
  layer_lstm(units = 16)
```

- Multiple inputs are at some point concatenated via a merge operation such as add (`layer_add`) or concatenate (`layer_concatenate`).

```
# concatenate operation
concatenated <- layer_concatenate(list(encoded_text, encoded_question))
answer <- concatenated %>%
  layer_dense(units = answer_vocabulary_size, activation = "softmax")

# model
model <- keras_model(list(text_input, question_input), answer)
```

- You can feed the model:
  - a list of arrays as inputs,
  - a dictionary that maps input names to arrays (if name was given to the inputs).

```
# feed list of inputs
model %>% fit(
  list(text, question), answers,
  epochs = 10, batch_size = 128
)

# feed named list of inputs
model %>% fit(
  list(text = text, question = question), answers,
  epochs = 10, batch_size = 128
)
```