Lab assignment 2: Linguistic processing using deep networks

By Ben Harvey

In these exercises, you will explore questions of language processing using the Keras library for RStudio. These follow from the exercises in Lab Assignment 1, so it is important that you complete those before starting. This should mean that you already have Keras and RStudio installed.

You should work in pairs on these exercises. We suggest doing these exercises on both of your computers simultaneously: this improves (student) learning and also makes it easier to find trivial mistakes.

In the following text, explanation is plain text, instructions are underlined, questions to answer are labelled, and RStudio console commands are written in `different font`. You are the first class doing this lab, so please note any mistakes and how you fixed them, as this will help our course development.

Exercise one: Text topic classification

Some linguistic analyses can be very straightforward, and do not require particularly complex network architectures. In this exercise, we will classify the topics of articles of news articles from the Reuters newswire database. These are short sections from the start of text articles, each labelled with a topic.

Launch RStudio and load Keras.

First, download the database, keeping only the 1000 most common words and separating 20% of the articles out to form a test set, as follows:

```
reuters <- dataset_reuters(num_words = 1000, test_split = 0.2)
```

Now extract training set inputs (x_train) and labels (y_train) from reuters$train$x and reuters$train$y respectively. Do the same for test set inputs and labels. (You have done very similar steps at the start of most Exercises in Assignment 1. You are just making new variables with shorter names than the existing structure fields.)

How many text sequences are in the test set and the training set? (Question 1)

Now make a variable (num_classes) giving the number of possible labels that the articles are divided into. This is simply the largest number in the labels, plus one (because zero is also used as a label). How many topic labels are included in this set? (Question 2)

Use this to number to convert y_train and y_test to categorical variables.
What code did you use for these two steps? (Question 3).

Now we will convert our text word sequences to sequences of numbers, as words are not suitable network input (numbers are). This process is called tokenizing, and

assigns each unique word as a number. After this, we will convert each word to a sparse matrix, with a 1 at the position of the tokenized word, and zeros everywhere else. Convert x_train and x_test into tokenized sequences as follows:

```
tokenizer <- text_tokenizer(num_words = 1000)
x_train <- sequences_to_matrix(tokenizer, x_train, mode =
'binary')
x_test <- sequences_to_matrix(tokenizer, x_test, mode =
'binary')
```

Now define a model, in this case a one-layer fully connected network. Start with a fully connected layer of 512 units and an input shape of 1000 (the size of each sparse tokenized word input). Use Relu activation. Follow this by a 50% dropout layer. Feed this into an output layer with num_classes units and softmax activation. What code did you use to define this model? (Question 4)

Now compile the model as in previous exercises. Use the loss function 'categorical_crossentropy'. Use the Adam optimization algorithm (optimizer = 'adam').

Adam (for 'adaptive moment estimation') is a recently developed optimization algorithm for stochastic optimization. The authors list the following attractive benefits of this algorithm:

- Straightforward to implement.
- Computationally efficient.
- Little memory requirements.
- Invariant to diagonal rescale of the gradients.
- Well suited for problems that are large in terms of data and/or parameters.
- Appropriate for non-stationary objectives.
- Appropriate for problems with very noisy/or sparse gradients.
- Hyper-parameters have intuitive interpretation and typically require little tuning.

Now fit the model using a batch size of 32, 5 epochs and a validation split of 10%. What code did you use for this fitting? (Question 5)

What accuracy do you find for the train set and validation set? What does this tell you about the generalisation of the model? (Question 6)

Considering the number of possible topic labels, how good do you think this classification accuracy is? (Question 7)

This network architecture is particularly simple. The 1000 most common words are fed into a single fully-connected layer of 512 units, then directly into the output labels layer. Considering this simple structure, explain how you think this network is performing the topic classification task (Question 8). For example, what type of feature in the input is being used to determine the topic?

Exercise two: Text sentiment analysis

We will now perform a sentiment analysis on a database of text reviews from the Internet Movie Database (IMDB).

We will use a 1-dimensional convolutional network, which has much in common with the networks you used in Assignment 1, so you can see how procedures change in linguistic analysis.

Data preparation:

Download the IMDB dataset that you will analyse, limiting this to the 5000 most common words:

```
imdb <- dataset_imdb(num_words = 5000)
```

For efficiency, each of these unique words is stored as an integer. These are stored in a matrix where each line (row) is a review, and each column is a word. Each of these reviews is labelled as either positive of negative (stored as 0 or 1). In the convolutional stage, we will use three-element filters, so that patterns of neighbouring words can be found together. To stop this filter size causing problems at the first and last word of each review, let's pad the reviews with zeros. We will also cut each review at 400 words, so they all have the same length.

Run:

```
x_train <- imdb$train$x %>%
  pad_sequences(maxlen = 400)
x_test <- imdb$test$x %>%
  pad_sequences(maxlen = 400)
```

Also define y_train as imdb$train$y and y_test as imdb$test$y.

The first stage of our model will use 'embedding'. Embedding is a common step in language processing using machine learning, as it compresses our 5000 words by representing each word as a position in a lower-dimensional space. Words with similar meanings and uses are grouped together in this space, effectively converting labels (words) into a semantic space. It is important that you understand this concept well. Research the concept of embedding layers online, and write a short explanation of the principle and why it is useful (Question 9).

Model definition:

Now we will define our model layers. We will begin by embedding our 5000 unique words into a 50-dimensional space. We will follow this by a dropout layer with a rate of 20%. We will then learn 250 x 1-dimensional convolutional filters, each with a length of 3 words. This is followed by max pooling over all convolutional units to find the one filter that best matches the input. This is then passed to a fully-connected layer with 250 units, to capture any pattern among the activity of the 250 convolutional filters, which passes into another layer with a dropout rate of 20%. Finally, these fully-connected units feed into a single output unit that classifies the review as negative or positive. This is activated by a sigmoid function to binarize the output unit's activation to zero or one. Define this model using the following code:

```
model <- keras_model_sequential()

model %>%
 layer_embedding(5000, 50, input_length = 400) %>%
  layer_dropout(0.2) %>%

  layer_conv_1d(250, 3, activation = "relu") %>%

  layer_global_max_pooling_1d() %>%
  layer_dense(250) %>%
  layer_dropout(0.2) %>%
  layer_activation("relu") %>%

  layer_dense(1) %>%
  layer_activation("sigmoid")
```

We will use a special loss function for binary outputs, and an optimisation function that is well suited to finding the best model parameters in this situation. Define this as follows:

```
model %>% compile(
  loss = "binary_crossentropy",
  optimizer = "adam",
  metrics = "accuracy"
)
```

Now run the model (like in Assignment 1) with a batch size of 32 for 4 epochs. Base your code on Assignment 1. What code did you use to run the model? (Question 10).

How did model performance improve with across epochs for both the test set and training set? What does this tell us about the generalisation of the resulting model? (Question 11).

Exercise 3: Text comprehension

… is difficult. Read the manuscript here, which describes a recent approach to using machine learning networks for text comprehension, and evaluation of performance in a standardised way: https://arxiv.org/pdf/1502.05698.pdf

Install the 'readr' package for Rstudio:

```
install.packages("readr")
```

Now implement one of the networks described in the paper, by working through the code at the following link. Do this first for QA1 questions, then for QA2 questions, both with 10,000 samples, by editing the 'challenges' section. Try to understand what each step in the code is achieving.

https://keras.rstudio.com/articles/examples/babi_memnn.html

Recurrent networks

Recurrent networks provide an excellent approach to processing time series inputs like text. These allow a model unit to be activated depending on its past activation state, so allow words some time back to determine the interpretation of the current word. This is particularly necessary for the correct interpretation of natural language.

Now implement a recurrent network for text comprehension, also described in the paper, by working through the code at the following link. (Note that the initial steps of both approaches are shared, until the 'data preparation' section).
Again, do this first for QA1 questions, then for QA2 questions, both with 10,000 samples, by editing the 'challenges' section. Try to understand what each step in the code is achieving.
https://keras.rstudio.com/articles/examples/babi_rnn.html

In about 1000 words (plus figures) explain how these two approaches differ, and compare their performance, generalisation and efficiency. (Question 12)