problems4

November 26, 2018

1 Problem sheet 4

1.1 1

In this problem, we generate Nietzsche text based on embedding words instead of characters. We adapt the `keras`-code from https://keras.rstudio.com/articles/examples/pretrained_word_embeddings.html.

Before getting started, think about possible advantages and disadvantages of using word embeddings

Next, download the Glove word vectors and create a look-up table.

```
In [ ]: emb_idx <- new.env(parent = emptyenv())
        lines <- readLines("./data/glove.6B.100d.txt")

        for (line in lines) {
          values <- strsplit(line, ' ', fixed = TRUE)[[1]]
          w <- values[[1]]
          coefs <- as.numeric(values[-1])
          emb_idx[[w]] <- coefs
        }
```

Proceed as in the lecture to load the Nietzsche text into a variable `text`. Then, tokenize the text.

```
In [ ]: tokenizer <- text_tokenizer()
        tokenizer %>% fit_text_tokenizer(text)
        t2s<- texts_to_sequences(tokenizer, text)[[1]]
```

Now, that the words are tokenized, we can set up the embedding matrix.

```
In [ ]: emb_dim <- 100
        w_idx <- tokenizer$word_index
        nwords <- length(w_idx) + 1

        emb_mat <- matrix(0L, nrow = nwords, ncol = emb_dim)
          for (w in names(w_idx)){
            idx <- w_idx[[w]]
            emb_vec <- emb_idx[[w]]
```

```
      # words not found in embedding index will be all-zeros.
      if (!is.null(emb_vec))
        emb_mat[index,] <- emb_vec
    }
```

The embedding matrix now becomes part of a an embedding layer.

```
In [ ]: emb_layer <- layer_embedding(
          input_dim = nwords,
          output_dim = emb_dim,
          weights = list(emb_mat),
          input_length = maxlen,
          trainable = F
        )
```

Build a keras model based on the embedding layer.

```
In [ ]: input <- layer_input(shape = list(maxlen),
                             dtype = 'int32')
        output <- ??

        model <- keras_model(input, output)
        model %>% compile(
          loss = "sparse_categorical_crossentropy",
          optimizer = optimizer_rmsprop(.01)
        )
```

Finally, we build the training data.

As in the lecture, chop the sequence into smaller bits and collect them in a variable `sentences`. Also store the next word into `next_words`.

```
In [ ]: sentences <- ??
        next_words <- ??
```

Fit the model!

1.2 2

Modify the weather example from the lecture to incorporate validation data.

1.3 3

Develop the seq2seq architecture for the weather example.

The idea is to use two neural networks, an **encoder** and a **decoder**. The encoder analyzes the history and summarizes it into a single vector. The decoder takes this vector and generates from it a target sequence.

https://arxiv.org/abs/1409.3215v3

In a wonderful blog post, François Chollet explains how to put the seq2seq-architecture into ‘keras. The corresponding pattern for R is described in https://github.com/rstudio/keras/issues/312. First, the encoder is constructed as

```
In [ ]: encoder_inputs  <- layer_input(shape = list(NULL, dim_data))
        encoder_results <- encoder_inputs %>%
                            layer_lstm(units = cell_size, return_state = T)
        encoder_states  <- encoder_results[2:3]
```

Second, the decoder reads as

```
In [ ]: decoder_inputs  <- layer_input(shape = list(NULL, dim_pred))

        decoder_lstm    <- layer_lstm(units = cell_size,
                                      return_sequences = T,
                                      return_state = T,
                                      stateful = F)
        decoder_results <- decoder_lstm(decoder_inputs, initial_state=encoder_states)
        decoder_dense   <- layer_dense(dim_pred, activation=?)
        decoder_outputs <- decoder_dense(decoder_results[[1]])
```

Now, adapt this idea to the weather data.