Deep Learning Lecture - Recurrent NN

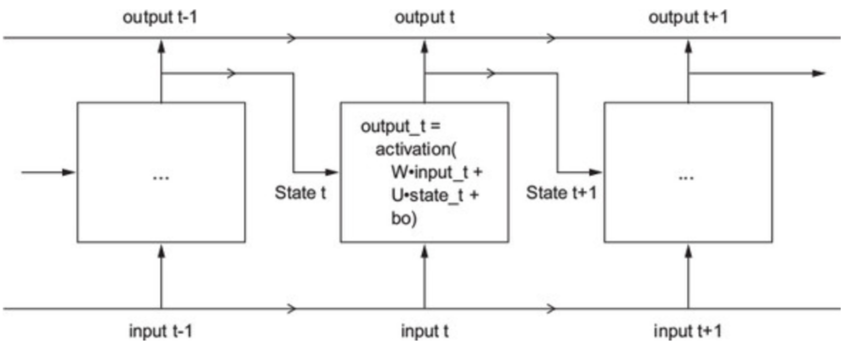
MA8701 General Statistical Methods

Thiago G. Martins, Department of Mathematical Sciences, NTNU

Spring 2019

Recurrent Neural Networks (RNNs)

A RNN process sequences by iterating through the sequence elements and maintaining a state containing information relative to what it has seen so far.



- The state of the RNN is reset between processing two different, independent sequences (such as two different IMDB reviews).

Transition equation for a simple RNN:

```
output_t <- tanh(as.numeric((W %*% input_t) + (U %*% state_t) + b))
```

Following is R pseudo-code for a simple RNN layer:

```
state_t <- 0
for (input_t in input_sequence) {
  output_t <- activation(dot(W, input_t) + dot(U, state_t) + b)
  state_t <- output_t
}
```

Embedding layer

Previously, we have encoded text data using integers.

- The order of the words were not preserved
- The representation did not captured any text semantics

Another approach to feed text to our models is to use an embedding layer:

- low-dimensional floating-point vectors learned from data.
- geometric relationships between word vectors should reflect the semantic relationships between these words.
- Take a 2D input tensor of integers of shape `(samples, sequence_length)`
- Return a 3D floating-point tensor of shape `(samples, sequence_length, embedding_dimensionality)`
- Such a 3D tensor can be processed by an RNN layer.
- The vector are randomly initialized prior to training.

The IMDB dataset

The objective here is to classify a movie review as either positive or negative.

Recurring neural networks with an embedding layer

- Data preparation parameters

```
max_features <- 10000 # Number of most frequent words
maxlen <- 500         # Padding the sequence of words to be of equal length
batch_size <- 32      # Batch size used for training
```

- Downloading the data

```
imdb <- dataset_imdb(num_words = max_features)
c(c(input_train, y_train), c(input_test, y_test)) %<-% imdb
cat(length(input_train), "train sequences\n")
```

```
## 25000 train sequences
```

```
cat(length(input_test), "test sequences")
```

```
## 25000 test sequences
```

- Padding the sequences

```
input_train <- pad_sequences(input_train, maxlen = maxlen)
input_test <- pad_sequences(input_test, maxlen = maxlen)
cat("input_train shape:", dim(input_train), "\n")
```

```
## input_train shape: 25000 500
```

```
cat("input_test shape:", dim(input_test), "\n")
```

```
## input_test shape: 25000 500
```

- Defining the model with embedding and simple RNN layers:

```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_simple_rnn(units = 32) %>%
  layer_dense(units = 1, activation = "sigmoid")
```

```
model
```

```
## Model
## ___________________________________________________________________________
## Layer (type)                     Output Shape                  Param #
## ===========================================================================
## embedding_1 (Embedding)          (None, None, 32)              320000
## ___________________________________________________________________________
## simple_rnn_1 (SimpleRNN)         (None, 32)                    2080
## ___________________________________________________________________________
## dense_1 (Dense)                  (None, 1)                     33
## ===========================================================================
## Total params: 322,113
## Trainable params: 322,113
## Non-trainable params: 0
## ___________________________________________________________________________
```

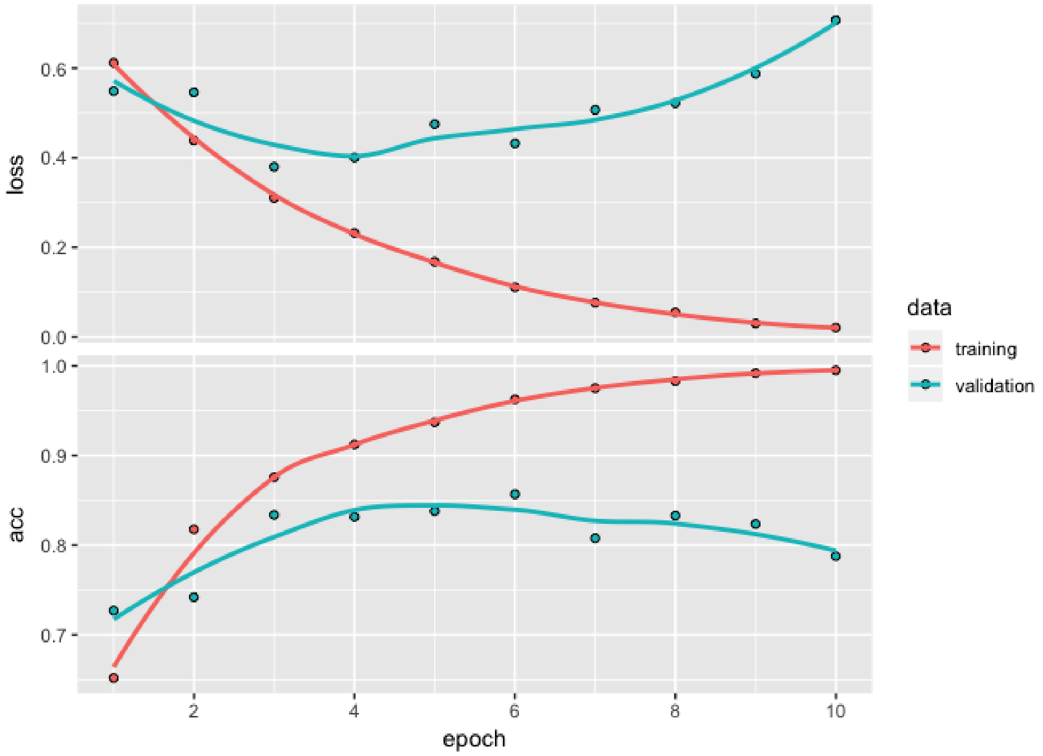
- Compiling the model

```
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
)
```

- Training and validation

```
history <- model %>% fit(
  input_train, y_train,
  epochs = 10,
  batch_size = 128,
  validation_split = 0.2
)
```

```
plot(history)
```



LSTM layer

The simple RNN layer should theoretically be able to retain at time `t` information about inputs seen many timesteps before.

- But in practice, such long-term dependencies are very hard to learn.

The LSTM (long-short term memory) layer was designed to address this issue.

- It allow past information to be reinjected at a later time.

Following is R pseudo-code for a LSTM layer:

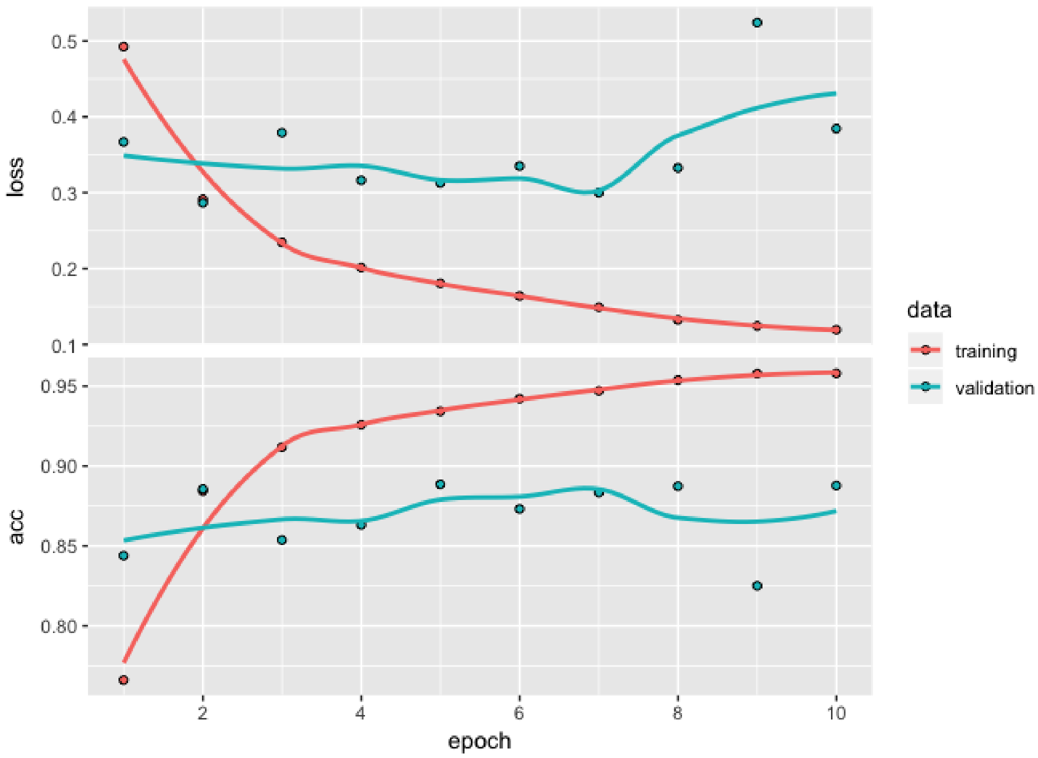
```
i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)

c_t+1 = i_t * k_t + c_t * f_t

output_t = activation(dot(state_t, Uo) + dot(input_t, Wo) + dot(C_t, Vo) + bo)
```

Using the LSTM layer in Keras:

```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = max_features, output_dim = 32) %>%
  layer_lstm(units = 32) %>%
  layer_dense(units = 1, activation = "sigmoid")
model %>% compile(
  optimizer = "rmsprop",
  loss = "binary_crossentropy",
  metrics = c("acc")
```

```r
)
history <- model %>% fit(
  input_train, y_train,
  epochs = 10,
  batch_size = 128,
  validation_split = 0.2
)
```

```r
plot(history)
```



Stacking recurrent layers

- We need to get all of the intermediate layers to return full sequences

```r
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = 10000, output_dim = 32) %>%
  layer_simple_rnn(units = 32, return_sequences = TRUE) %>%
  layer_simple_rnn(units = 32, return_sequences = TRUE) %>%
  layer_simple_rnn(units = 32, return_sequences = TRUE) %>%
  layer_simple_rnn(units = 32)                                1
```