Deep Learning Lecture 1 - Reuters: Densily connected NN

MA8701 General Statistical Methods

Thiago G. Martins, Department of Mathematical Sciences, NTNU

Spring 2019

The Reuters dataset

The objective here is to classify short news stories into one of 46 topics available.

Preparing the data

Here, we use the multi-assignment operator (`%<-%`) from the `zeallot` package to unpack the list into a set of distinct variables.

```
reuters <- dataset_reuters(num_words = 10000)
c(c(train_data, train_labels), c(test_data, test_labels)) %<-% reuters
```

```
length(train_data)
```

```
## [1] 8982
```

```
length(test_data)
```

```
## [1] 2246
```

As with the IMDB reviews, each example is a list of integers (word indices):

```
train_data[[1]]
```

```
##  [1]    1    2    2    8   43   10  447    5   25  207  270    5 3095  111
## [15]   16  369  186   90   67    7   89    5   19  102    6   19  124   15
## [29]   90   67   84   22  482   26    7   48    4   49    8  864   39  209
## [43]  154    6  151    6   83   11   15   22  155   11   15    7   48    9
## [57] 4579 1005  504    6  258    6  272   11   15   22  134   44   11   15
## [71]   16    8  197 1245   90   67   52   29  209   30   32  132    6  109
## [85]   15   17   12
```

```
train_labels[[1]]
```

```
## [1] 3
```

You can vectorize the data with the exact same code as in the IMDB example

```
vectorize_sequences <- function(sequences, dimension = 10000) {
  results <- matrix(0, nrow = length(sequences), ncol = dimension)
  for (i in 1:length(sequences))
    results[i, sequences[[i]]] <- 1
  results
}

x_train <- vectorize_sequences(train_data)
x_test <- vectorize_sequences(test_data)
```

Vectorize the labels:

```
one_hot_train_labels <- to_categorical(train_labels)
one_hot_test_labels <- to_categorical(test_labels)
```

Building the model

- The dimensionality of the output space (46 classes) is much larger.

Information bottleneck

- Each layer can only access information present in the output of the previous layer.
- Each layer can potentially become an information bottleneck.
- A 16-dimensional intermediate layer may be too limited to learn to separate 46 different classes:
- Such small layers may act as information bottlenecks, permanently dropping relevant information.

For this reason we will use larger layers. Let's go with 64 units.

```
model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = c(10000)) %>%
```

```
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 46, activation = "softmax")
```

Compiling the model

The best loss function to use in this case is `categorical_crossentropy`.

```
model %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)
```

Validating your approach

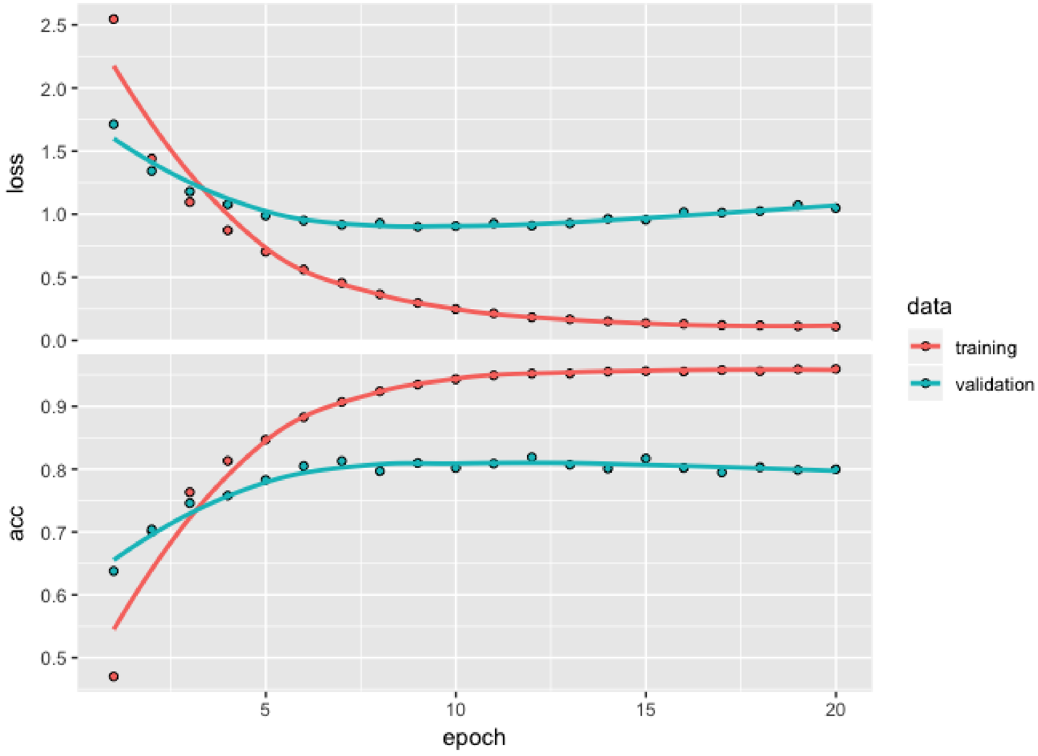
Let's set apart 1000 samples in the training data to use as a validation set.

```
val_indices <- 1:1000

x_val <- x_train[val_indices,]
partial_x_train <- x_train[-val_indices,]

y_val <- one_hot_train_labels[val_indices,]
partial_y_train = one_hot_train_labels[-val_indices,]
```

Now, let's train the network for 20 epochs.

```
history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 20,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
```

```
plot(history)
```



The network begins to overfit after nine epochs. Let's train a new network from scratch for nine epochs and then evaluate it on the test set.

```
model <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu", input_shape = c(10000)) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 46, activation = "softmax")

model %>% compile(
  optimizer = "rmsprop",
  loss = "categorical_crossentropy",
  metrics = c("accuracy")
)

history <- model %>% fit(
  partial_x_train,
  partial_y_train,
  epochs = 9,
  batch_size = 512,
  validation_data = list(x_val, y_val)
)
```

```
results <- model %>% evaluate(x_test, one_hot_test_labels)

results
```

```
## $loss
## [1] 1.021877
##
## $acc
## [1] 0.777382
```

This approach reaches an accuracy of ~ 79%. With a balanced binary classification problem, the accuracy reached by a purely random classifier would be 50%. But in this case it's closer to 18%, so the results seem pretty good, at least when compared to a random baseline:

```
test_labels_copy <- test_labels
test_labels_copy <- sample(test_labels_copy)
length(which(test_labels == test_labels_copy)) / length(test_labels)
```

```
## [1] 0.1843277
```

Predictions on new data

```
predictions <- model %>% predict(x_test)
```

Each entry in predictions is a vector of length 46:

```
dim(predictions)
```

```
## [1] 2246   46
```

The coefficients in this vector sum to 1:

```
sum(predictions[1,])
```

```
## [1] 1
```

The largest entry is the predicted class—the class with the highest probability:

```
which.max(predictions[1,])
```

```
## [1] 4
```

Recommended exercise 2

Use a vector of integers as labels instead of the one-hot encoding used above. Remember that this choice will impact the loss function used to train the model.