Practice

Callbacks implementation

Data

Recent technological advances and international efforts, such as The Cancer Genome Atlas (TCGA), have made available several pan-cancer datasets encompassing multiple omics layers with detailed clinical information in large collection of samples. The need has thus arisen for the development of computational methods aimed at improving cancer subtyping and biomarker identification from multi-modal data.

In https://www.frontiersin.org/articles/10.3389/fonc.2020.01065/full three multi-modal cancer datasets generated by The Cancer Genome Atlas (TCGA) Research Network (https://www.cancer. gov/tcga) are considered. Protein expression (prot), gene expression (gene), and copy number variants (cnv) are used to predict breast invasive carcinoma (BRCA) estrogen receptor status (0: negative; 1: positive) (BRCA-ER, N = 381).

In this practice we will focus on protein- expression data. We show the callbacks implementation and propose to adapt the script to allow hiperparameter tuning with `tfruns()`.

```
library(keras)
```

```
mprotein<-read.csv("mprotein.csv",header = T)
```

```
set.seed(123)
training<-sample(1:nrow(mprotein),2*nrow(mprotein)/3)


xtrain<-mprotein[training,-c(1,2)]
xtest<-mprotein[-training,-c(1,2)]

xtrain<-scale(xtrain)
xtest<-scale(xtest)


ytrain<-mprotein[training,2]
ytest<-mprotein[-training,2]

ylabels<-vector()
ylabels[ytrain=="Positive"]<-1
ylabels[ytrain=="Negative"]<-0


ytestlabels<-vector()
ytestlabels[ytest=="Positive"]<-1
ytestlabels[ytest=="Negative"]<-0
```

```
FLAGS <- flags(
  flag_numeric("dfeat", 20)
)
```

```
dnn <- keras_model_sequential() %>%
  layer_dense(units=50,activation="relu",input_shape = c(142)) %>%
  layer_dense(units= FLAGS$dfeat,activation="relu")  %>%
  layer_dense(units= 1,activation="sigmoid")

summary(dnn)
```

```
## Model: "sequential"
## _________________________________________________________________________________
##  Layer (type)                       Output Shape                    Param #
## =================================================================================
##  dense_2 (Dense)                    (None, 50)                      7150
##
##  dense_1 (Dense)                    (None, 20)                      1020
##
##  dense (Dense)                      (None, 1)                       21
##
## =================================================================================
## Total params: 8,191
## Trainable params: 8,191
## Non-trainable params: 0
## _________________________________________________________________________________
```

```
dnn %>%compile(
  optimizer = "rmsprop",
  loss = 'binary_crossentropy',
  metric = "acc"
)
```

```
x<-as.matrix(xtrain)
```

```
dnn %>% fit(
  x,ylabels,
  epochs = 25,
  batch_size=64,
  validation_split = 0.2
  )
```

Keras callbacks

When you train a model, there are a lot of things you can't predict from the start. In in particular, one cannot know how many epochs it will take to obtain an optimal validation loss. The examples so far have adopted the training strategy for enough times that you start to overfit, using the first run to calculate the appropriate number of epochs for train and finally launch a new training from scratch using this number. Of course, this approach is wasteful.

A much better way to handle this is to stop training when it is measured that the loss of validation no longer improves. This can be achieved using a Keras `callback`. A `callback` is an object that is passed to the model on the call to adjust and is called by the model at various points during training. You have access to all available data on the status of the model and its performance, and you can take action: interrupt training, save a model, load a set of different weights, or alter the state of the model. Here are some examples of ways you can use `callbacks`:

Keras includes a number of built-in `callbacks` (this is not an exhaustive list): `callback_model_checkpoint()`, `callback_early_stopping()`, `callback_learning_rate_scheduler()`, `callback_reduce_lr_on_plateau()` and `callback_csv_logger()`.

```
callbacks_list<-list(
  callback_early_stopping(
    monitor = "acc",
    patience=8
  ),
  callback_model_checkpoint(
    filepath = "mymodels.h5",
    monitor = "acc",
    save_best_only = T
  ),
  callback_reduce_lr_on_plateau(
    monitor="acc",
    factor = 0.1,
    patience = 2
  )
)
```

```
dnn %>% fit(
  x,ylabels,
  epochs = 10, batch_size = 32,callbacks = callbacks_list
)
```

```
dnn %>%
  evaluate(as.matrix(xtest), ytestlabels)
```

```
##      loss       acc
## 0.4380070 0.8976378
```