Introduction to Keras

Saint Louis R User Group

Joshua Ulrich

2017-11-09

Disclaimer

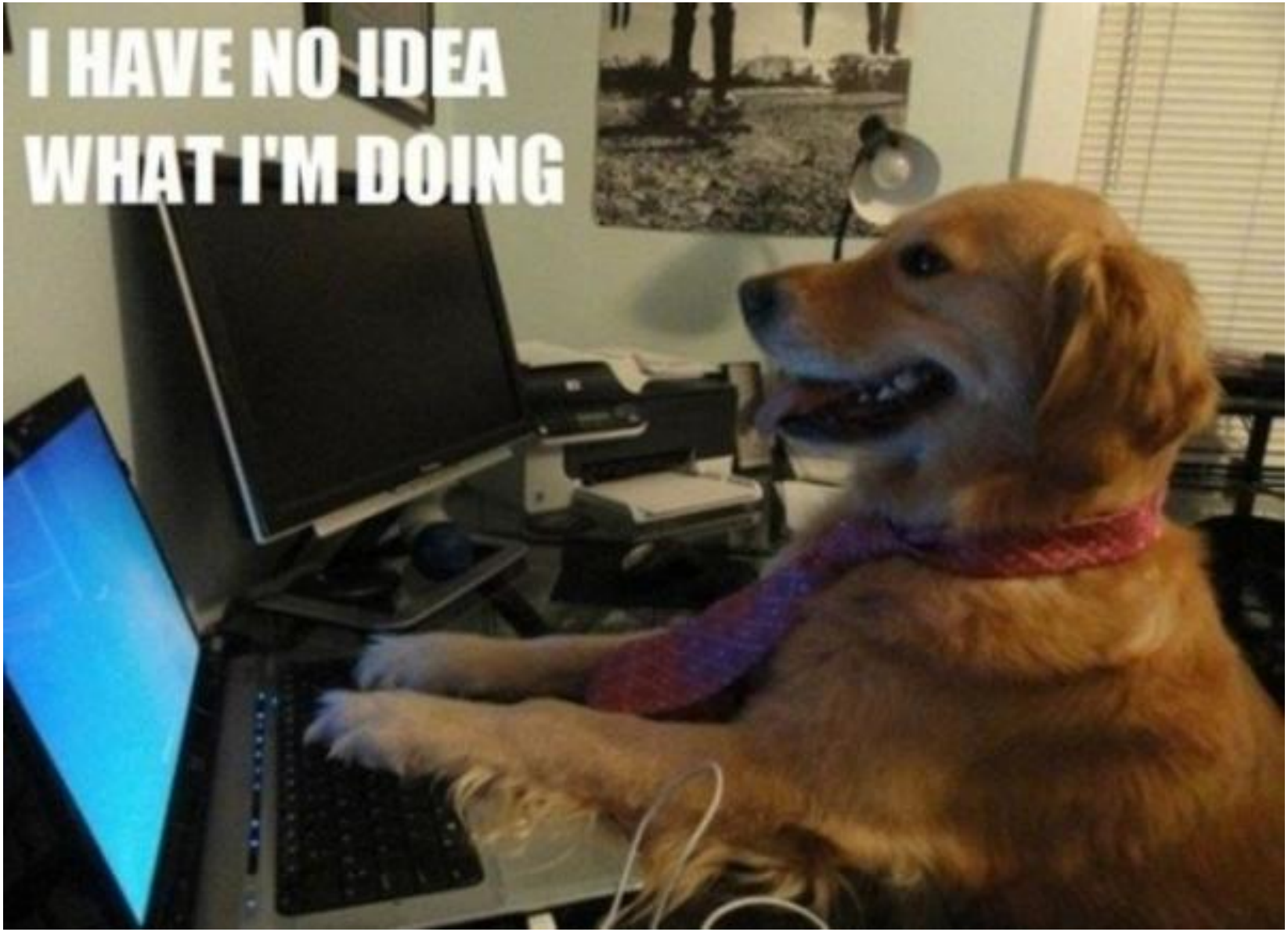


Figure 1:

What is Keras?

- High-level neural networks API
- Focus on prototyping and fast experimentation via
  - user friendliness
  - modularity
  - extensibility
- Can run on CPU or GPU
- Can run on top of TensorFlow, CNTK, or Theano
- Supports convolutional and recurrent networks

Why deep learning?

- It's the new hotness.

Installing Keras

Installation is easy, if you use the CPU. Installation using one or more GPUs is more involved. We'll install using the CPU, which is the default.

```
R> install.packages("keras")
R> keras::install_keras()
```

Installing Keras

Okay, so maybe it's not *that* easy...

```
R> install.packages("keras")
R> keras::install_keras()
Error: Prerequisites for installing TensorFlow
  not available.

Execute the following at a terminal to install
the prerequisites:

$ sudo apt-get install python-pip python-virtualenv

R> q("no")
```

Installing Keras

. . . but the error gave us the command we need to run in order to continue the installation.

I'm using Ubuntu, but the errors on other operating systems are just as helpful.

So let's run that command at the terminal.

```
> sudo apt-get install python-pip python-virtualenv
```

Installing Keras

Now, back to R

```
R> keras::install_keras()
...
Installation of TensorFlow complete.


Installation of Keras complete.

R>
```

Sweet! Let's run a simple example to make sure everything is working.

Example

We can learn the basics of Keras by walking through a simple example: recognizing handwritten digits from the MNIST dataset. MNIST consists of 28 x 28 grayscale images of handwritten digits like:

Figure 2:

Example

```
library(keras)
mnist <- dataset_mnist()
x_train <- mnist$train$x
y_train <- mnist$train$y
x_test <- mnist$test$x
y_test <- mnist$test$y

n_train <- nrow(x_train)
n_test <- nrow(x_test)
```

Reshape the data

R stores arrays in column-major order, while Keras stores them in row-major order.

```
# reshape
x_train <- array_reshape(x_train, c(n_train, 784))
Error in array_reshape(x_train, c(n_train, 784)) :
  could not find function "array_reshape"
```

Whoops, you need the latest development version (for `reticulate::array_reshape`) in order to run the example on https://keras.rstudio.com

Installing development versions

You can use:

```
remotes::install_github("rstudio/keras")
# or
devtools::install_github("rstudio/keras")
```

These commands currently (re-)install (development versions?) all dependencies. Building some packages (e.g. Rcpp) from source can take a bit of time.

Installing development versions

I'll install the packages individually instead.

```
git clone https://github.com/rstudio/reticulate.git
git clone https://github.com/rstudio/keras.git
git clone https://github.com/rstudio/tensorflow.git
Rscript -e 'install.packages("zeallot")'
R CMD build reticulate keras tensorflow
R CMD INSTALL reticulate_* keras_* tensorflow_*
```

Okay, back to the example...

Data munging

```
# reshape (array_reshape in reticulate > 1.2 only)
x_train <- array_reshape(x_train, c(n_train, 784))
x_test <- array_reshape(x_test, c(n_test, 784))

# rescale
x_train <- x_train / 255
x_test <- x_test / 255

# one-hot encode
y_train <- to_categorical(y_train, 10)
y_test <- to_categorical(y_test, 10)
```

Two ways to define models

Define an 'empty' model and add layers one at a time.

```
model <- keras_model_sequential()
model %>%
  layer_dense(units = 256, activation = 'relu',
              input_shape = 784) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 128, activation = 'relu') %>%
  layer_dropout(rate = 0.3) %>%
  layer_dense(units = 10, activation = 'softmax')
```

Two ways to define models

Specify the layers in the model constructor. You can add more later, if you want.

```
model <- keras_model_sequential(
  layers = list(
    layer_dense(NULL, 256, 'relu', input_shape = 784),
    layer_dropout(rate = 0.4),
    layer_dense(NULL, 128, 'relu'),
    layer_dropout(rate = 0.3),
    layer_dense(NULL, 10, 'softmax')
  )
)
```

Summary of model details

```
summary(model)
```

```
## _________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =================================================================
## dense_1 (Dense)              (None, 256)               200960
## _________________________________________________________________
## dropout_1 (Dropout)          (None, 256)               0
## _________________________________________________________________
## dense_2 (Dense)              (None, 128)               32896
## _________________________________________________________________
## dropout_2 (Dropout)          (None, 128)               0
## _________________________________________________________________
## dense_3 (Dense)              (None, 10)                1290
## =================================================================
## Total params: 235,146
## Trainable params: 235,146
```

Compile the model

Need to configure the learning process before training the model. This is done via `compile()`, which has 3 primary arguments:

- objective function (the function you're optimizing)
- optimizer (method used to find the function minimum)
- metrics (what are evaluated during training & testing)

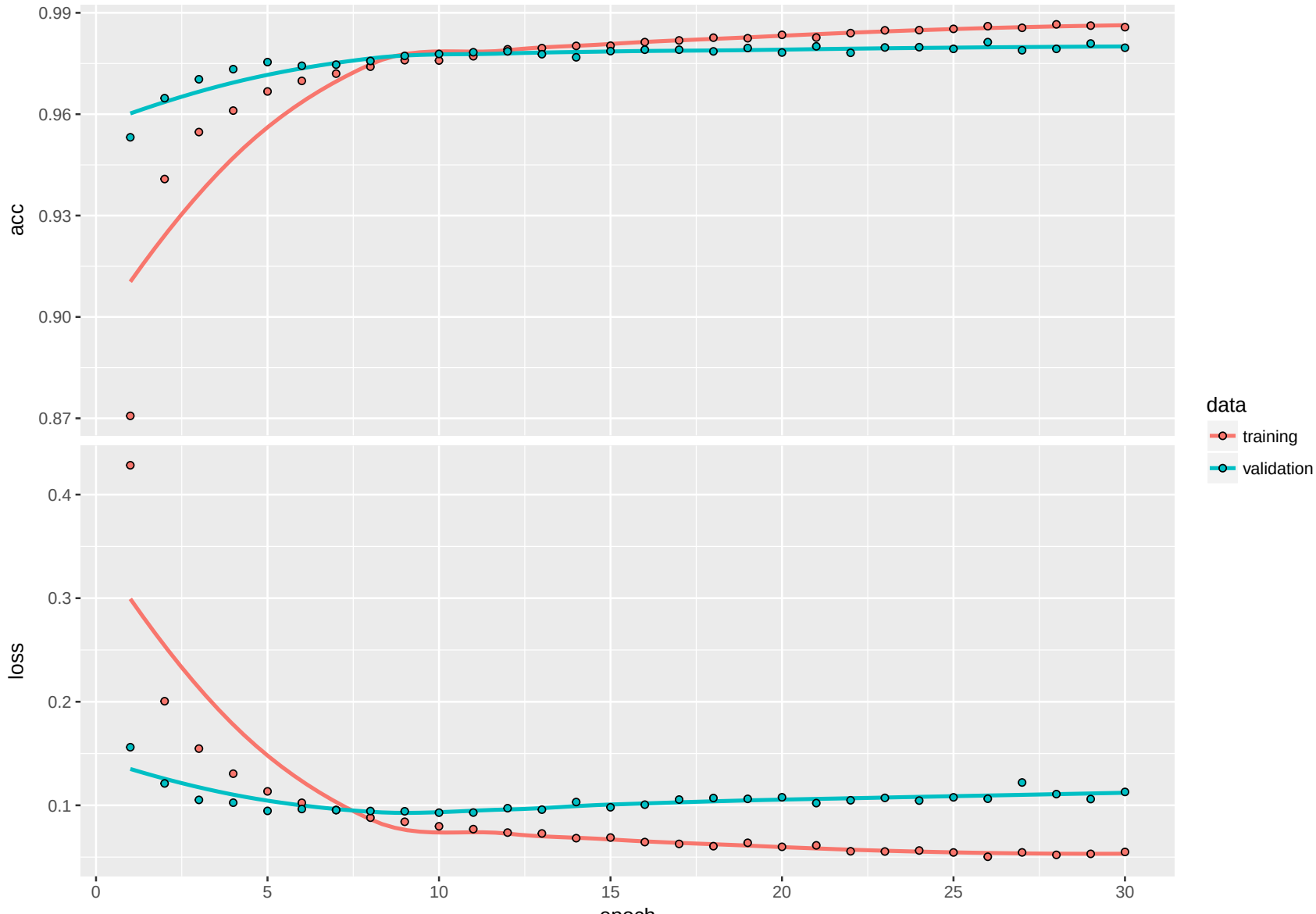
```
compile(model,
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
```

Fitting / Training

```
# takes a few minutes on a modest CPU--16 threads
fit_model <- fit(
  model, x_train, y_train,
  epochs = 30, batch_size = 128,
  validation_split = 0.2
)
```

Plot fit results

```
plot(fit_model)
```



Evaluate model performance

```
evaluate(model, x_test, y_test)
```

```
## $loss
## [1] 0.1075985
##
## $acc
## [1] 0.9801
```

generate predictions

```
predictions <- predict_classes(model, x_test)
head(predictions)
```

```
## [1] 7 2 1 0 4 1
```