VAE_MNIST_2.R

esteban.vegas

2022-05-12

```
# VAE_MNIST_2
#
# Create a function to customize the VAE loss. It is a diferent solution to VAE_MNIST_1
```

This script demonstrates how to build a variational autoencoder with Keras and deconvolution layers. Reference: "Auto-Encoding Variational Bayes" https://arxiv.org/abs/1312.6114

```
# Note: This code reflects pre-TF2 idioms.
# For an example of a TF2-style modularized VAE, see e.g.: https://github.com/rstudio/keras/blob/master/
# Also cf. the tfprobability-style of coding VAEs: https://rstudio.github.io/tfprobability/

# With TF-2, you can still run this code due to the following line:
if (tensorflow::tf$executing_eagerly())
  tensorflow::tf$compat$v1$disable_eager_execution()
```

```
## Loaded Tensorflow version 2.4.1
```

```
library(keras)
K <- keras::backend()

#### Parameterization ####

# input image dimensions
img_rows <- 28L
img_cols <- 28L
# color channels (1 = grayscale, 3 = RGB)
img_chns <- 1L

# number of convolutional filters to use
filters <- 64L

# convolution kernel size
num_conv <- 3L

latent_dim <- 2L
intermediate_dim <- 128L
epsilon_std <- 1.0

# training parameters
batch_size <- 100L
epochs <- 5L
```

```r
#### Model Construction ####

original_img_size <- c(img_rows, img_cols, img_chns)

x <- layer_input(shape = c(original_img_size))

conv_1 <- layer_conv_2d(
  x,
  filters = img_chns,
  kernel_size = c(2L, 2L),
  strides = c(1L, 1L),
  padding = "same",
  activation = "relu"
)

conv_2 <- layer_conv_2d(
  conv_1,
  filters = filters,
  kernel_size = c(2L, 2L),
  strides = c(2L, 2L),
  padding = "same",
  activation = "relu"
)

conv_3 <- layer_conv_2d(
  conv_2,
  filters = filters,
  kernel_size = c(num_conv, num_conv),
  strides = c(1L, 1L),
  padding = "same",
  activation = "relu"
)

conv_4 <- layer_conv_2d(
  conv_3,
  filters = filters,
  kernel_size = c(num_conv, num_conv),
  strides = c(1L, 1L),
  padding = "same",
  activation = "relu"
)

flat <- layer_flatten(conv_4)
hidden <- layer_dense(flat, units = intermediate_dim, activation = "relu")

z_mean <- layer_dense(hidden, units = latent_dim)
z_log_var <- layer_dense(hidden, units = latent_dim)

sampling <- function(args) {
  z_mean <- args[, 1:(latent_dim)]
  z_log_var <- args[, (latent_dim + 1):(2 * latent_dim)]
```

```
  epsilon <- k_random_normal(
    shape = c(k_shape(z_mean)[[1]]),
    mean = 0.,
    stddev = epsilon_std
  )
  z_mean + k_exp(z_log_var) * epsilon
}

z <- layer_concatenate(list(z_mean, z_log_var)) %>% layer_lambda(sampling)

output_shape <- c(batch_size, 14L, 14L, filters)

decoder_hidden <- layer_dense(units = intermediate_dim, activation = "relu")
decoder_upsample <- layer_dense(units = prod(output_shape[-1]), activation = "relu")

decoder_reshape <- layer_reshape(target_shape = output_shape[-1])
decoder_deconv_1 <- layer_conv_2d_transpose(
  filters = filters,
  kernel_size = c(num_conv, num_conv),
  strides = c(1L, 1L),
  padding = "same",
  activation = "relu"
)

decoder_deconv_2 <- layer_conv_2d_transpose(
  filters = filters,
  kernel_size = c(num_conv, num_conv),
  strides = c(1L, 1L),
  padding = "same",
  activation = "relu"
)

decoder_deconv_3_upsample <- layer_conv_2d_transpose(
  filters = filters,
  kernel_size = c(3L, 3L),
  strides = c(2L, 2L),
  padding = "valid",
  activation = "relu"
)

decoder_mean_squash <- layer_conv_2d(
  filters = img_chns,
  kernel_size = c(2L, 2L),
  strides = c(1L, 1L),
  padding = "valid",
  activation = "sigmoid"
)

hidden_decoded <- decoder_hidden(z)
up_decoded <- decoder_upsample(hidden_decoded)
reshape_decoded <- decoder_reshape(up_decoded)
deconv_1_decoded <- decoder_deconv_1(reshape_decoded)
deconv_2_decoded <- decoder_deconv_2(deconv_1_decoded)
```

```r
x_decoded_relu <- decoder_deconv_3_upsample(deconv_2_decoded)
x_decoded_mean_squash <- decoder_mean_squash(x_decoded_relu)

# custom loss function
vae_loss <- function(x, x_decoded_mean_squash) {
  x <- k_flatten(x)
  x_decoded_mean_squash <- k_flatten(x_decoded_mean_squash)
  xent_loss <- 1.0 * img_rows * img_cols *
    loss_binary_crossentropy(x, x_decoded_mean_squash)
  kl_loss <- -0.5 * k_mean(1 + z_log_var - k_square(z_mean) -
                             k_exp(z_log_var), axis = -1L)
  k_mean(xent_loss + kl_loss)
}

## variational autoencoder
vae <- keras_model(x, x_decoded_mean_squash)
vae %>% compile(optimizer = "rmsprop", loss = vae_loss)
summary(vae)
```

```
## Model: "model"
## ________________________________________________________________________________
## Layer (type)               Output Shape      Param #  Connected to
## ================================================================================
## input_1 (InputLayer)       [(None, 28, 28, 1 0
## ________________________________________________________________________________
## conv2d (Conv2D)            (None, 28, 28, 1) 5        input_1[0][0]
## ________________________________________________________________________________
## conv2d_1 (Conv2D)          (None, 14, 14, 64 320      conv2d[0][0]
## ________________________________________________________________________________
## conv2d_2 (Conv2D)          (None, 14, 14, 64 36928    conv2d_1[0][0]
## ________________________________________________________________________________
## conv2d_3 (Conv2D)          (None, 14, 14, 64 36928    conv2d_2[0][0]
## ________________________________________________________________________________
## flatten (Flatten)          (None, 12544)     0        conv2d_3[0][0]
## ________________________________________________________________________________
## dense (Dense)              (None, 128)       1605760  flatten[0][0]
## ________________________________________________________________________________
## dense_1 (Dense)            (None, 2)         258      dense[0][0]
## ________________________________________________________________________________
## dense_2 (Dense)            (None, 2)         258      dense[0][0]
## ________________________________________________________________________________
## concatenate (Concatenate)  (None, 4)         0        dense_1[0][0]
##                                                       dense_2[0][0]
## ________________________________________________________________________________
## lambda (Lambda)            (None, 2)         0        concatenate[0][0]
## ________________________________________________________________________________
## dense_3 (Dense)            (None, 128)       384      lambda[0][0]
## ________________________________________________________________________________
## dense_4 (Dense)            (None, 12544)     1618176  dense_3[0][0]
## ________________________________________________________________________________
## reshape (Reshape)          (None, 14, 14, 64 0        dense_4[0][0]
## ________________________________________________________________________________
## conv2d_transpose (Conv2DT (None, 14, 14, 64 36928    reshape[0][0]
```

```
## ________________________________________________________________________________
## conv2d_transpose_1 (Conv2 (None, 14, 14, 64 36928    conv2d_transpose[0][0]
## ________________________________________________________________________________
## conv2d_transpose_2 (Conv2 (None, 29, 29, 64 36928    conv2d_transpose_1[0][0]
## ________________________________________________________________________________
## conv2d_4 (Conv2D)         (None, 28, 28, 1) 257      conv2d_transpose_2[0][0]
## ================================================================================
## Total params: 3,410,058
## Trainable params: 3,410,058
## Non-trainable params: 0
## ________________________________________________________________________________
```

```r
## encoder: model to project inputs on the latent space
encoder <- keras_model(x, z_mean)

## build a digit generator that can sample from the learned distribution
gen_decoder_input <- layer_input(shape = latent_dim)
gen_hidden_decoded <- decoder_hidden(gen_decoder_input)
gen_up_decoded <- decoder_upsample(gen_hidden_decoded)
gen_reshape_decoded <- decoder_reshape(gen_up_decoded)
gen_deconv_1_decoded <- decoder_deconv_1(gen_reshape_decoded)
gen_deconv_2_decoded <- decoder_deconv_2(gen_deconv_1_decoded)
gen_x_decoded_relu <- decoder_deconv_3_upsample(gen_deconv_2_decoded)
gen_x_decoded_mean_squash <- decoder_mean_squash(gen_x_decoded_relu)
generator <- keras_model(gen_decoder_input, gen_x_decoded_mean_squash)


#### Data Preparation ####

mnist <- dataset_mnist()
data <- lapply(mnist, function(m) {
  array_reshape(m$x / 255, dim = c(dim(m$x)[1], original_img_size))
})
x_train <- data$train
x_test <- data$test


#### Model Fitting ####

vae %>% fit(
  x_train, x_train,
  shuffle = TRUE,
  epochs = epochs,
  batch_size = batch_size,
  validation_data = list(x_test, x_test)
)


#### Visualizations ####

library(ggplot2)
library(dplyr)
```

```
##
```

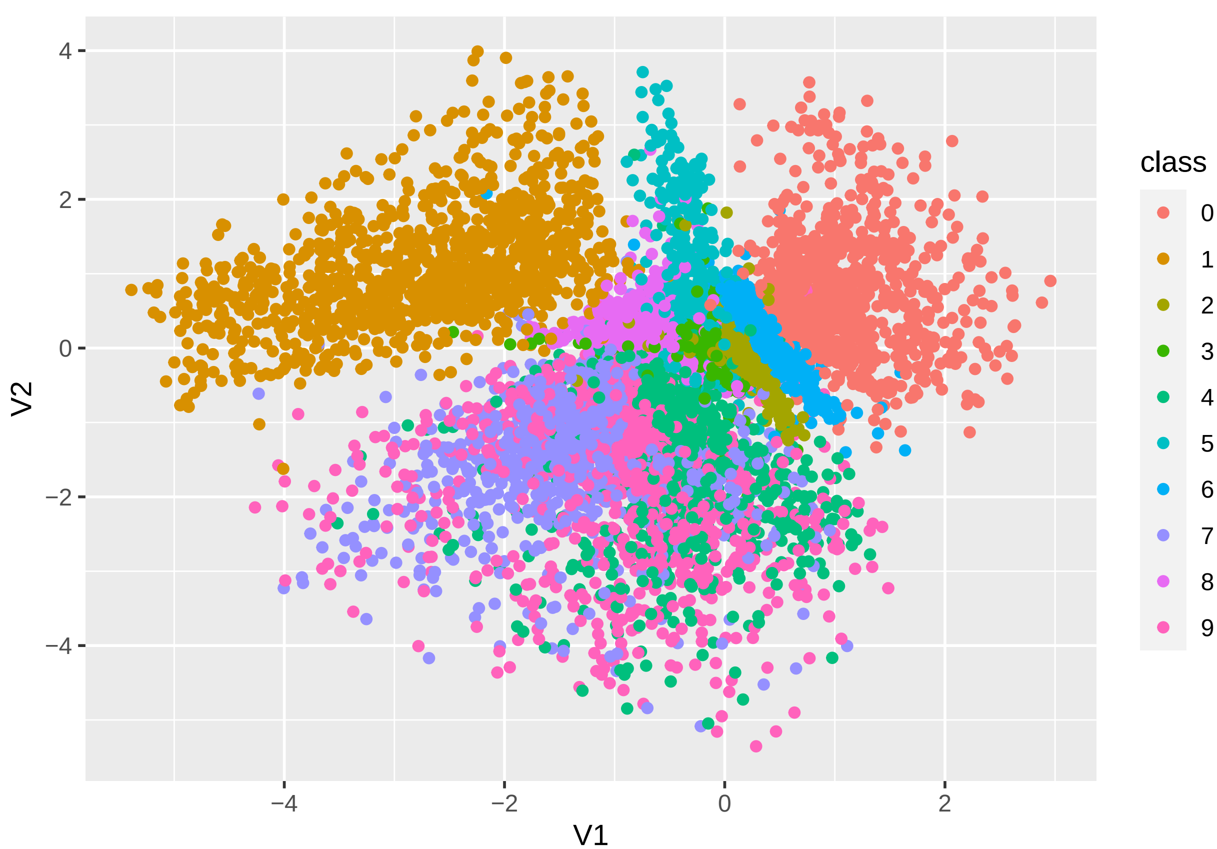
```
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
## display a 2D plot of the digit classes in the latent space
x_test_encoded <- predict(encoder, x_test, batch_size = batch_size)
x_test_encoded %>%
  as_tibble() %>%
  mutate(class = as.factor(mnist$test$y)) %>%
  ggplot(aes(x = V1, y = V2, colour = class)) + geom_point()
```

```
## Warning: The `x` argument of `as_tibble.matrix()` must have unique column names if `.name_repair` is
## Using compatibility `.name_repair`.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```



```r
## display a 2D manifold of the digits
n <- 15  # figure with 15x15 digits
digit_size <- 28

# we will sample n points within [-4, 4] standard deviations
grid_x <- seq(-4, 4, length.out = n)
grid_y <- seq(-4, 4, length.out = n)

rows <- NULL
for(i in 1:length(grid_x)){
  column <- NULL
  for(j in 1:length(grid_y)){
    z_sample <- matrix(c(grid_x[i], grid_y[j]), ncol = 2)
    column <- rbind(column, predict(generator, z_sample) %>% matrix(ncol = digit_size))
  }
  rows <- cbind(rows, column)
}
rows %>% as.raster() %>% plot()
```