VAE_MNIST_1.R

esteban.vegas

2022-05-10

```r
# VAE_MNIST_1
#
# Create CustomVariationalLayer to run a customize the VAE loss




## 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
```

```r
library(keras)

img_shape <- c(28, 28, 1)
batch_size <- 16
latent_dim <- 2L  # Dimensionality of the latent space: a plane

input_img <- layer_input(shape = img_shape)

x <- input_img %>%
  layer_conv_2d(filters = 32, kernel_size = 3, padding = "same",
                activation = "relu") %>%
  layer_conv_2d(filters = 64, kernel_size = 3, padding = "same",
                activation = "relu", strides = c(2, 2)) %>%
  layer_conv_2d(filters = 64, kernel_size = 3, padding = "same",
                activation = "relu") %>%
  layer_conv_2d(filters = 64, kernel_size = 3, padding = "same",
                activation = "relu")

shape_before_flattening <- k_int_shape(x) #

x <- x %>%
  layer_flatten() %>%
  layer_dense(units = 32, activation = "relu")

z_mean <- x %>%
  layer_dense(units = latent_dim)

z_log_var <- x %>%
```

```r
  layer_dense(units = latent_dim)



# Sampling function
sampling <- function(args) {
  c(z_mean, z_log_var) %<-% args
  epsilon <- k_random_normal(shape = list(k_shape(z_mean)[1], latent_dim),
                             mean = 0, stddev = 1)
  z_mean + k_exp(z_log_var) * epsilon
}

# Point randomly sampled
z <- list(z_mean, z_log_var) %>%
  layer_lambda(sampling)



# This is the input where we will feed `z`.
decoder_input <- layer_input(k_int_shape(z)[-1])

x <- decoder_input %>%
  # Upsample to the correct number of units
  layer_dense(units = prod(as.integer(shape_before_flattening[-1])),
              activation = "relu") %>%
  # Reshapes into an image of the same shape as before the last flatten layer
  layer_reshape(target_shape = shape_before_flattening[-1]) %>%
  # Applies and then reverses the operation to the initial stack of
  # convolution layers
  layer_conv_2d_transpose(filters = 32, kernel_size = 3, padding = "same",
                          activation = "relu", strides = c(2, 2)) %>%
  layer_conv_2d(filters = 1, kernel_size = 3, padding = "same",
                activation = "sigmoid")
  # We end up with a feature map of the same size as the original input.

# This is our decoder model.
decoder <- keras_model(decoder_input, x)

# We then apply it to `z` to recover the decoded `z`.
z_decoded <- decoder(z)



library(R6)

CustomVariationalLayer <- R6Class("CustomVariationalLayer",

  inherit = KerasLayer,

  public = list(

    vae_loss = function(x, z_decoded) {
      x <- k_flatten(x)
```

```
      z_decoded <- k_flatten(z_decoded)
      xent_loss <- metric_binary_crossentropy(x, z_decoded)
      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 + 1e-3 * kl_loss)
    },

    call = function(inputs, mask = NULL) {
      x <- inputs[[1]]
      z_decoded <- inputs[[2]]
      loss <- self$vae_loss(x, z_decoded)
      self$add_loss(loss, inputs = inputs)
      x
    }
  )
)

layer_variational <- function(object) {
  create_layer(CustomVariationalLayer, object, list())
}

# Call the custom layer on the input and the decoded output to obtain
# the final model output
y <- list(input_img, z_decoded) %>%
  layer_variational()


# VAE model
vae <- keras_model(input_img, y)

vae %>% compile(
  optimizer = "rmsprop",
  loss = NULL
)

# Trains the VAE on MNIST digits
mnist <- dataset_mnist()
c(c(x_train, y_train), c(x_test, y_test)) %<-% mnist

x_train <- x_train / 255
x_train <- array_reshape(x_train, dim =c(dim(x_train), 1))

x_test <- x_test / 255
x_test <- array_reshape(x_test, dim =c(dim(x_test), 1))

vae %>% fit(
  x = x_train, y = NULL,
  epochs = 10,
  batch_size = batch_size,
  validation_data = list(x_test, NULL)
)
```

```r
# Exploration latent dimension

n <- 15             # Number of rows / columns of digits
digit_size <- 28    # Height / width of digits in pixels

# Transforms linearly spaced coordinates on the unit square through the inverse
# CDF (ppf) of the Gaussian to produce values of the latent variables z,
# because the prior of the latent space is Gaussian
grid_x <- qnorm(seq(0.05, 0.95, length.out = n))
grid_y <- qnorm(seq(0.05, 0.95, length.out = n))

op <- par(mfrow = c(n, n), mar = c(0,0,0,0), bg = "black")
for (i in 1:length(grid_x)) {
  yi <- grid_x[[i]]
  for (j in 1:length(grid_y)) {
    xi <- grid_y[[j]]
    z_sample <- matrix(c(xi, yi), nrow = 1, ncol = 2)
    z_sample <- t(replicate(batch_size, z_sample, simplify = "matrix"))
    x_decoded <- decoder %>% predict(z_sample, batch_size = batch_size)
    digit <- array_reshape(x_decoded[1,,,], dim = c(digit_size, digit_size))
    plot(as.raster(digit))
  }
}
```