MINST VAE

Bruce Campbell

5/7/2018

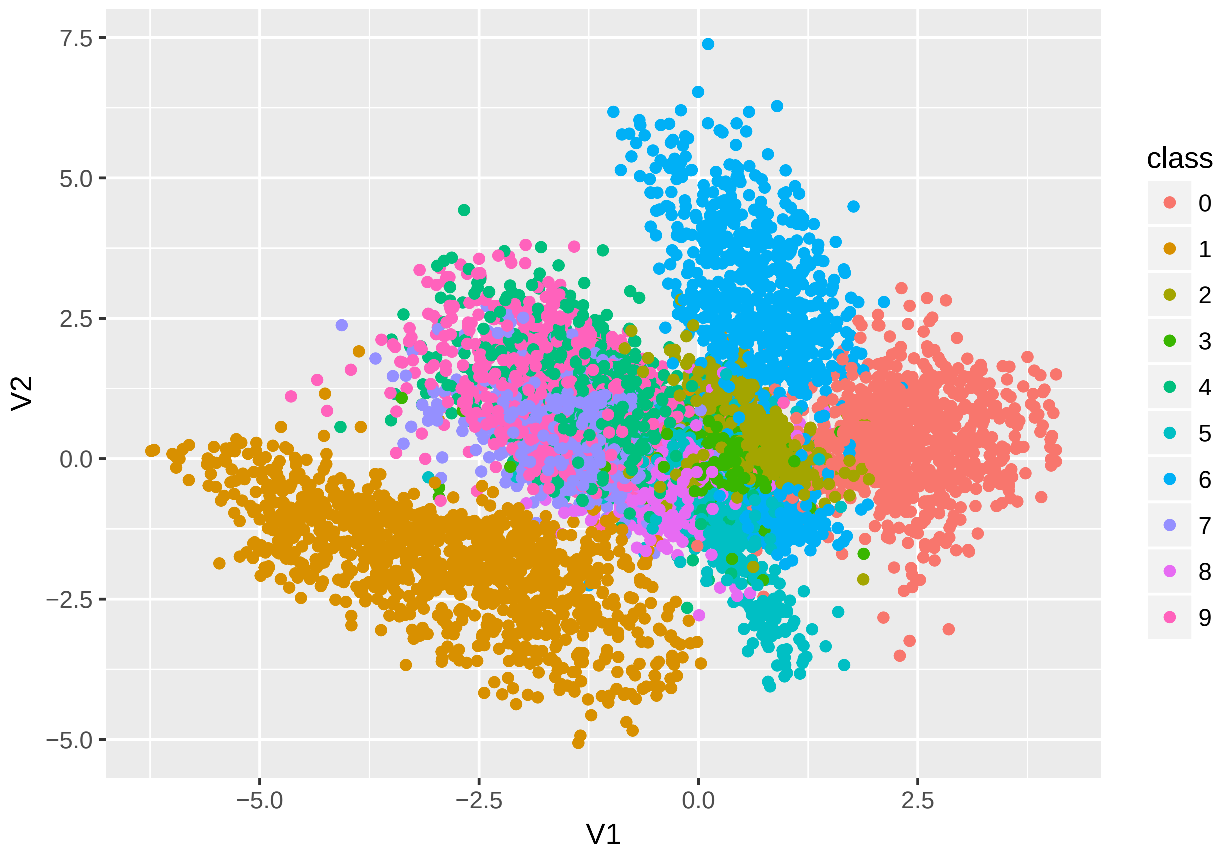
```r
library(keras)
K <- keras::backend()

# Parameters --------------------------------------------------------------

batch_size <- 100L
original_dim <- 784L
latent_dim <- 2L
intermediate_dim <- 256L
epochs <- 50L
epsilon_std <- 1.0

# Model definition --------------------------------------------------------

x <- layer_input(shape = c(original_dim))
h <- layer_dense(x, intermediate_dim, activation = "relu")
z_mean <- layer_dense(h, latent_dim)
z_log_var <- layer_dense(h, latent_dim)

sampling <- function(arg){
  z_mean <- arg[, 1:(latent_dim)]
  z_log_var <- arg[, (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/2)*epsilon
}

# note that "output_shape" isn't necessary with the TensorFlow backend
z <- layer_concatenate(list(z_mean, z_log_var)) %>%
  layer_lambda(sampling)

# we instantiate these layers separately so as to reuse them later
decoder_h <- layer_dense(units = intermediate_dim, activation = "relu")
decoder_mean <- layer_dense(units = original_dim, activation = "sigmoid")
h_decoded <- decoder_h(z)
x_decoded_mean <- decoder_mean(h_decoded)

# end-to-end autoencoder
vae <- keras_model(x, x_decoded_mean)

# encoder, from inputs to latent space
encoder <- keras_model(x, z_mean)
```

```r
# generator, from latent space to reconstructed inputs
decoder_input <- layer_input(shape = latent_dim)
h_decoded_2 <- decoder_h(decoder_input)
x_decoded_mean_2 <- decoder_mean(h_decoded_2)
generator <- keras_model(decoder_input, x_decoded_mean_2)


vae_loss <- function(x, x_decoded_mean){
  xent_loss <- (original_dim/1.0)*loss_binary_crossentropy(x, x_decoded_mean)
  kl_loss <- -0.5*k_mean(1 + z_log_var - k_square(z_mean) - k_exp(z_log_var), axis = -1L)
  xent_loss + kl_loss
}

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


# Data preparation --------------------------------------------------

mnist <- dataset_mnist()
x_train <- mnist$train$x/255
x_test <- mnist$test$x/255
x_train <- x_train %>% apply(1, as.numeric) %>% t()
x_test <- x_test %>% apply(1, as.numeric) %>% t()


# Model training ----------------------------------------------------

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
x_test_encoded <- predict(encoder, x_test, batch_size = batch_size)

x_test_encoded %>%
```

```r
  as_data_frame() %>%
  mutate(class = as.factor(mnist$test$y)) %>%
  ggplot(aes(x = V1, y = V2, colour = class)) + geom_point()
```




```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 = 28) )
  }
  rows <- cbind(rows, column)
}
rows %>% as.raster() %>% plot()
```