Lab 12

Emilio Dorigatti

2021-02-05

```
library(tensorflow)
cat(tf$version$GIT_VERSION)
```

Exercise 1

https://github.com/soumith/ganhacks

```
library(keras)

mnist = dataset_mnist()
x_train = (mnist$train$x - 127.5) / 127.5
y_train = to_categorical(mnist$train$y)
input_size = 28 * 28
dim(x_train) <- c(nrow(x_train), input_size)
```

```
optim = optimizer_adam(lr=0.0002, beta_1=0.5)
noise_size = 100

generator <- keras_model_sequential() %>%
  layer_dense(256, input_shape = c(as.integer(noise_size))) %>%
  layer_activation_leaky_relu(alpha = 0.2) %>%
  layer_dense(512) %>%
  layer_activation_leaky_relu(alpha = 0.2) %>%
  layer_dense(1024) %>%
  layer_activation_leaky_relu(alpha = 0.2) %>%
  layer_dense(784, activation = "tanh")

generator %>% compile(
  loss = "binary_crossentropy",
  optimizer = optim,
)
```

```
discriminator <- keras_model_sequential() %>%
  layer_dense(1024, input_shape = c(as.integer(784))) %>%
  layer_activation_leaky_relu(alpha = 0.2) %>%
  layer_dropout(0.3) %>%
  layer_dense(512) %>%
  layer_activation_leaky_relu(alpha = 0.2) %>%
  layer_dropout(0.3) %>%
  layer_dense(256) %>%
  layer_activation_leaky_relu(alpha = 0.2) %>%
  layer_dropout(0.3) %>%
  layer_dense(1, activation = "sigmoid")

discriminator %>% compile(
  loss = "binary_crossentropy",
  optimizer = optim,
)
```

```
gan_input = layer_input(shape = c(as.integer(noise_size)))
gan_output = discriminator(generator(gan_input))
gan = keras_model(gan_input, gan_output)
gan %>% compile(loss = "binary_crossentropy", optimizer = optim)
```

```
batch_size = 64

generator_losses = c()
discriminator_losses = c()

for(i in 1:10000) {
  #!hwbegin TODO get a random batch of images
  batch_idx = sample(1:nrow(x_train), batch_size)
  batch_images = x_train[batch_idx,]
  #!hwend

  #!hwbegin TODO create a matrix of normally-distributed noise
  noise = matrix(rnorm(batch_size * noise_size), nrow = batch_size)
  #!hwend

  #!hwbegin TODO use the generator to generate images from the noise
  generated = generator$predict(noise)
  #!hwend

  #!hwbegin TODO concatenate the batches of real and generated images
  batch_x = rbind(batch_images, generated)
  #!hwend

  #!hwbegin TODO create labels for the batch, using 0.9 for real images and 0.1 for generated images
  batch_y = c(rep(0.9, batch_size), rep(0.1, batch_size))
  #!hwend

  #!hwbegin TODO unfreeze the weights of the discriminator
  discriminator$trainable = TRUE
  #!hwend

  dloss = (
    #!hwbegin TODO perform a step of SGD on the discriminator
    discriminator$train_on_batch(batch_x, batch_y)
    #!hwend
  )

  #!hwbegin TODO freeze the weights of the discriminator
  discriminator$trainable = FALSE
  #!hwend

  gloss = (
    #!hwbegin TODO perform a step of SGD on the entire GAN using noise as input
    gan$train_on_batch(
      matrix(rnorm(batch_size * noise_size), nrow = batch_size),
      rep(1, batch_size)
    )
    #!hwend
  )

  # store losses
  discriminator_losses = c(discriminator_losses, dloss)
```

```
  generator_losses = c(generator_losses, gloss)

  # print progress message
  if(i %% 1000 == 0) {
    cat(
      "===  batch", i,
      "\n     generator loss",
      mean(generator_losses[length(generator_losses)-99:length(generator_losses)]),
      "\n     discriminator loss",
      mean(discriminator_losses[length(discriminator_losses)-99:length(discriminator_losses)]),
      "\n"
    )
  }
}
```

```
library(ggplot2)
ggplot() +
  geom_line(aes(x=1:length(generator_losses), y=generator_losses, color="r")) +
  geom_line(aes(x=1:length(discriminator_losses), y=discriminator_losses, color="b"))
```

```
noise = matrix(rnorm(batch_size * noise_size), nrow = batch_size)
generated = generator$predict(noise)
```

```
library(grid)

lay <- grid.layout(3, 3)
vplay <- viewport(layout=lay)
pushViewport(vplay)

for(i in 0:8) {
  gen = matrix(generated[i + 1,] / 2 + 0.5, nrow = 28)
  gen[gen < 0] = 0
  gen[gen > 1] = 1

  pushViewport(viewport(layout.pos.row=as.integer(i / 3) + 1, layout.pos.col=i %% 3 + 1))
  grid.raster(gen, interpolate = FALSE)
  upViewport()
}
```