hy-23 Created using Colaboratory History

1 contributor

561 lines (561 sloc) | 57.3 KB

Open in Colab

In [1]:
```
import tensorflow as tf
import tensorflow.keras as k
import numpy as np
import matplotlib.pyplot as plt
```

In [2]:
```
# seed
np.random.seed(11)
tf.random.set_seed(11)

# hyperparameters
batch_size = 256
max_epochs = 50
lr = 1e-3
momentum = 8e-1
hidden_dim = 128
original_dim = 784
```

In [3]:
```
(x_train, _), (x_test, _) = k.datasets.mnist.load_data()
```

In [4]:
```
x_train = x_train / 255.0
x_test  = x_test / 255.0

x_train = x_train.astype(np.float32)
x_test = x_test.astype(np.float32)

x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1]*x_train.shape[2]))
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1]*x_test.shape[2]))
```

In [5]:
```
training_dataset = tf.data.Dataset.from_tensor_slices(x_train).batch(batch_size)
```

AutoEncoder Class

In [6]:
```
class Encoder(k.layers.Layer):
  def __init__(self, hidden_dim):
    super(Encoder, self).__init__()
    self.hidden_layer = k.layers.Dense(units=hidden_dim, activation='relu')

  def call(self, input_features):
    activation = self.hidden_layer(input_features)
    return activation

class Decoder(k.layers.Layer):
  def __init__(self, original_dim):
    super(Decoder, self).__init__()
    self.output_layer = k.layers.Dense(units=original_dim, activation = 'relu')

  def call(self, encoded_features):
    activation = self.output_layer(encoded_features)
    return activation

class Autoencoder(k.Model):
  def __init__(self, hidden_dim, original_dim):
    super(Autoencoder, self).__init__()
    # initialize the loss to empty list.
    self.loss = []
    # instantiate encoder and decoder objects.
    # this is object composition as learnt in Desing Theory.
    self.encoder = Encoder(hidden_dim=hidden_dim)
    self.decoder = Decoder(original_dim=original_dim)

  def call(self, input_features):
    #print("inside call of Autoencoder")
    encoded = self.encoder(input_features)
    decoded = self.decoder(encoded)
    return decoded
```

Use Autoencoder class

In [7]:

```
autoencoder = Autoencoder(hidden_dim=hidden_dim, original_dim=original_dim)
optimizer = k.optimizers.Adam(learning_rate=lr)

def cost_function(real, pred):
  return tf.reduce_mean(tf.square(tf.subtract(real, pred)))

def train(loss, model, optimizer, org):
  with tf.GradientTape() as tape:
    #print("batch is of type {}".format(type(org)))
    pred  = model(org) # return value is (batch_size, 784) decode
    #print("shape of pred {}".format(pred.shape))
    error = loss(org, pred)
    gradients = tape.gradient(error, model.trainable_variables)
    gradient_variables = zip(gradients, model.trainable_variables)
  optimizer.apply_gradients(gradient_variables)
  return error

def train_loop(loss, model, optimizer, dataset, epochs=20):
  for epoch in range(epochs):
    epoch_loss = 0
    for step, batch_features in enumerate(dataset):
      error = train(loss, model, optimizer, batch_features)
      epoch_loss += error
    model.loss.append(epoch_loss)
    print('Epoch {}/{}. Loss: {}'.format(epoch+1, epochs, epoch_loss.numpy()))
```

In [8]:

```
train_loop(cost_function, autoencoder, optimizer, training_dataset, epochs=max_epochs)
```

```
Epoch 1/50. Loss: 6.228333950042725
Epoch 2/50. Loss: 2.239034652709961
Epoch 3/50. Loss: 1.7826430797576904
Epoch 4/50. Loss: 1.5789159536361694
Epoch 5/50. Loss: 1.4672006368637085
Epoch 6/50. Loss: 1.3902620077133179
Epoch 7/50. Loss: 1.3327252864837646
Epoch 8/50. Loss: 1.2876464128494263
Epoch 9/50. Loss: 1.2564762830734253
Epoch 10/50. Loss: 1.2367315292358398
Epoch 11/50. Loss: 1.2224229574203491
Epoch 12/50. Loss: 1.2074127197265625
Epoch 13/50. Loss: 1.1942561864852905
Epoch 14/50. Loss: 1.1829586029052734
Epoch 15/50. Loss: 1.1746004819869995
Epoch 16/50. Loss: 1.1688674688339233
Epoch 17/50. Loss: 1.164124608039856
Epoch 18/50. Loss: 1.1577184200286865
Epoch 19/50. Loss: 1.1483063697814941
Epoch 20/50. Loss: 1.1436494588851929
Epoch 21/50. Loss: 1.1403621435165405
Epoch 22/50. Loss: 1.13750159740448
Epoch 23/50. Loss: 1.1350390911102295
Epoch 24/50. Loss: 1.1327935457229614
Epoch 25/50. Loss: 1.1307650804519653
Epoch 26/50. Loss: 1.1289427280426025
Epoch 27/50. Loss: 1.1271594762802124
Epoch 28/50. Loss: 1.1255664825439453
Epoch 29/50. Loss: 1.1241445541381836
Epoch 30/50. Loss: 1.1228108406066895
Epoch 31/50. Loss: 1.1215500831604004
Epoch 32/50. Loss: 1.120348334312439
Epoch 33/50. Loss: 1.1192383766174316
Epoch 34/50. Loss: 1.1181410551071167
Epoch 35/50. Loss: 1.1171300411224365
Epoch 36/50. Loss: 1.1161916255950928
Epoch 37/50. Loss: 1.1152658462524414
Epoch 38/50. Loss: 1.114380121231079
Epoch 39/50. Loss: 1.1135915517807007
Epoch 40/50. Loss: 1.1128453016281128
Epoch 41/50. Loss: 1.1121163368225098
Epoch 42/50. Loss: 1.1114715337753296
Epoch 43/50. Loss: 1.1108250617980957
Epoch 44/50. Loss: 1.1102039813995361
Epoch 45/50. Loss: 1.1095937490463257
Epoch 46/50. Loss: 1.1090441942214966
Epoch 47/50. Loss: 1.1085431575775146
Epoch 48/50. Loss: 1.1080554723739624
Epoch 49/50. Loss: 1.1075657606124878
Epoch 50/50. Loss: 1.1070923805236816
```

Sequential Model

In [9]:
```
in_shape = x_train.shape[1:]
sModel = k.models.Sequential()
sModel.add(k.layers.Dense(units=hidden_dim, activation='relu',
                          input_shape=in_shape))
sModel.add(k.layers.Dense(units=original_dim, activation='relu'))
```

In [10]:
```
sModel.compile(optimizer=optimizer, loss='mean_squared_error')
```

In [11]:
```
history = sModel.fit(x=x_train, y=x_train, batch_size=256, epochs=max_epochs, verbose = 0)
```

In [12]:
```
sModel.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_2 (Dense)             (None, 128)               100480

 dense_3 (Dense)             (None, 784)               101136

=================================================================
Total params: 201,616
Trainable params: 201,616
Non-trainable params: 0
_________________________________________________________________
```

In [13]:
```
autoencoder.summary()
```

```
Model: "autoencoder"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 encoder (Encoder)           multiple                  100480

 decoder (Decoder)           multiple                  101136

=================================================================
Total params: 201,616
Trainable params: 201,616
Non-trainable params: 0
_________________________________________________________________
```

In [14]:
```
x_train[1, :].shape
```

Out[14]:
```
(784,)
```

In [15]:
```
sample = np.expand_dims(x_train[4958], axis=0)
```

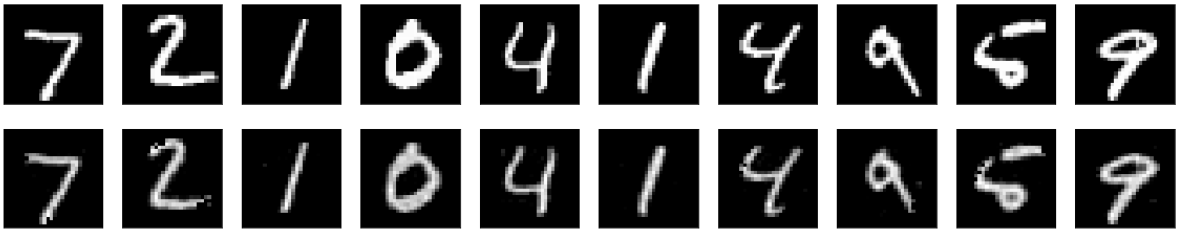
In [16]:
```
seq_pred = sModel.predict(x_train)
aut_pred = autoencoder(x_train)
```

In [17]:
```
print("shape of seq_pred: {}".format(seq_pred.shape))
print("shape of aut_pred: {}".format(aut_pred.shape))

s = tf.reduce_sum(seq_pred - sample)
a = tf.reduce_sum(aut_pred - sample)

print("autoencoder error: {}".format(a))
print("sequential  error: {}".format(s))
```

```
shape of seq_pred: (60000, 784)
shape of aut_pred: (60000, 784)
autoencoder error: 2149299.0
sequential  error: 2139589.0
```

In [18]:
```
number = 10
plt.figure(figsize=(20,4))
for index in range(number):
  # display original
  ax = plt.subplot(2, number, index + 1)
  plt.imshow(x_test[index].reshape(28, 28), cmap='gray')
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
```

```
  # display reconstruction
  ax = plt.subplot(2, number, index + 1 + number)
  plt.imshow(autoencoder(x_test)[index].numpy().reshape(28, 28),
             cmap='gray')
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```



In [19]:

```
number = 10
plt.figure(figsize=(20,4))
for index in range(number):
  # display original
  ax = plt.subplot(2, number, index + 1)
  plt.imshow(x_test[index].reshape(28, 28), cmap='gray')
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)

  # display reconstruction
  ax = plt.subplot(2, number, index + 1 + number)
  plt.imshow(sModel.predict(x_test)[index].reshape(28, 28),
             cmap='gray')
  ax.get_xaxis().set_visible(False)
  ax.get_yaxis().set_visible(False)
plt.show()
```