Master Informatique EID2

Deep Learning

TP 4 : Apprentissage profond pour la classification et la transformation d'espace

Mohamed Ben Saad
Elias ABDELLI

La classification avec l’apprentissage profond

```
#@title Importation Library
import  numpy as np
import matplotlib.pyplot as plt
import sklearn.datasets
from sklearn import decomposition
from tensorflow import keras
from tensorflow.keras import layers
```

Importation Library

Model_exec

```
#@title Model_exec
def Model_exec(batch_size, nbr_neural, epochs, activation_function='relu', output_activation=False):
  inputs = keras.Input(shape=(784,), name='digits')
  x = layers.Dense(nbr_neural, activation=activation_function, name='dense_1')(inputs)
  if output_activation :
    outputs = layers.Dense(10,activation=output_activation, name='predictions')(x)
  else:
    outputs = layers.Dense(10, name='predictions')(x)
  model = keras.Model(inputs=inputs, outputs=outputs)
  (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
  x_train = x_train.reshape(60000, 784).astype('float32') / 255
  x_test = x_test.reshape(10000, 784).astype('float32') / 255
  y_train = y_train.astype('float32')
  y_test = y_test.astype('float32')
  x_val = x_train[-10000:]
  y_val = y_train[-10000:]
  x_train = x_train[:-10000]
  y_train = y_train[:-10000]
  model.compile(optimizer=keras.optimizers.RMSprop(learning_rate=1),  # Optimizer
              # Loss function to minimize
              loss=keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              # List of metrics to monitor
              metrics=['sparse_categorical_accuracy'])
  print('# Fit model on training data')
  history = model.fit(x_train, y_train,
                    batch_size=batch_size,
                    epochs=epochs,
                    validation_data=(x_val, y_val))
  print('history dict:', history.history)
```

Sigmoid 100 neurones

```
#@title Sigmoid 100 neurones
Model_exec(nbr_neural=100,batch_size=100,epochs=1, activation_function='sigmoid')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
# Fit model on training data
500/500 [==============================] - 2s 3ms/step - loss: 13.2863 - sparse_categorical_accuracy: 0.5317 - val_loss: 6.0941 - val_sparse_categorical_accuracy: 0.7026
```

Outputsoftmax 5 Iterations

```
#@title Outputsoftmax 5 Iterations
Model_exec(nbr_neural=20 ,batch_size=1000,epochs=5, output_Activation_function='softmax' , activation_function='sigmoid')
```

```
# Fit model on training data
Epoch 1/5
50/50 [==============================] - 0s 8ms/step - loss: 2.1987 - sparse_categorical_accuracy: 0.2568 - val_loss: 2.1683 - val_sparse_categorical_accuracy: 0.2863
Epoch 2/5
50/50 [==============================] - 0s 7ms/step - loss: 2.1687 - sparse_categorical_accuracy: 0.2899 - val_loss: 2.1663 - val_sparse_categorical_accuracy: 0.2941
Epoch 3/5
50/50 [==============================] - 0s 7ms/step - loss: 2.1657 - sparse_categorical_accuracy: 0.2940 - val_loss: 2.1684 - val_sparse_categorical_accuracy: 0.2916
Epoch 4/5
50/50 [==============================] - 0s 7ms/step - loss: 2.1638 - sparse_categorical_accuracy: 0.2968 - val_loss: 2.1704 - val_sparse_categorical_accuracy: 0.2906
Epoch 5/5
50/50 [==============================] - 0s 7ms/step - loss: 2.1619 - sparse_categorical_accuracy: 0.2983 - val_loss: 2.1658 - val_sparse_categorical_accuracy: 0.2950
```

Outputsoftmax 20 Iterations

```python
#@title Outputsoftmax 20 Iterations
Model_exec(nbr_neural=20 ,batch_size=1000,epochs=20, output_Activation_function='softmax' , activation_function='sigmoid')
```

```
# Fit model on training data
Epoch 1/20
50/50 [==============================] - 0s 9ms/step - loss: 2.2316 - sparse_categorical_accuracy: 0.2258 - val_loss: 2.1837 - val_sparse_categorical_accuracy: 0.2709
Epoch 2/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1892 - sparse_categorical_accuracy: 0.2700 - val_loss: 2.1879 - val_sparse_categorical_accuracy: 0.2708
Epoch 3/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1760 - sparse_categorical_accuracy: 0.2835 - val_loss: 2.1672 - val_sparse_categorical_accuracy: 0.2923
Epoch 4/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1712 - sparse_categorical_accuracy: 0.2887 - val_loss: 2.1709 - val_sparse_categorical_accuracy: 0.2900
Epoch 5/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1764 - sparse_categorical_accuracy: 0.2834 - val_loss: 2.1746 - val_sparse_categorical_accuracy: 0.2863
Epoch 6/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1739 - sparse_categorical_accuracy: 0.2861 - val_loss: 2.1727 - val_sparse_categorical_accuracy: 0.2860
Epoch 7/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1694 - sparse_categorical_accuracy: 0.2907 - val_loss: 2.1669 - val_sparse_categorical_accuracy: 0.2935
Epoch 8/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1687 - sparse_categorical_accuracy: 0.2914 - val_loss: 2.1666 - val_sparse_categorical_accuracy: 0.2944
Epoch 9/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1665 - sparse_categorical_accuracy: 0.2939 - val_loss: 2.1718 - val_sparse_categorical_accuracy: 0.2883
Epoch 10/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1690 - sparse_categorical_accuracy: 0.2913 - val_loss: 2.1952 - val_sparse_categorical_accuracy: 0.2655
Epoch 11/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1717 - sparse_categorical_accuracy: 0.2889 - val_loss: 2.1773 - val_sparse_categorical_accuracy: 0.2836
Epoch 12/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1639 - sparse_categorical_accuracy: 0.2967 - val_loss: 2.1710 - val_sparse_categorical_accuracy: 0.2902
Epoch 13/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1684 - sparse_categorical_accuracy: 0.2920 - val_loss: 2.1711 - val_sparse_categorical_accuracy: 0.2895
Epoch 14/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1668 - sparse_categorical_accuracy: 0.2939 - val_loss: 2.1763 - val_sparse_categorical_accuracy: 0.2831
Epoch 15/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1665 - sparse_categorical_accuracy: 0.2942 - val_loss: 2.1702 - val_sparse_categorical_accuracy: 0.2911
Epoch 16/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1654 - sparse_categorical_accuracy: 0.2955 - val_loss: 2.1709 - val_sparse_categorical_accuracy: 0.2899
Epoch 17/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1673 - sparse_categorical_accuracy: 0.2933 - val_loss: 2.1675 - val_sparse_categorical_accuracy: 0.2933
Epoch 18/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1651 - sparse_categorical_accuracy: 0.2954 - val_loss: 2.1748 - val_sparse_categorical_accuracy: 0.2862
Epoch 19/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1647 - sparse_categorical_accuracy: 0.2960 - val_loss: 2.1654 - val_sparse_categorical_accuracy: 0.2958
Epoch 20/20
50/50 [==============================] - 0s 7ms/step - loss: 2.1652 - sparse_categorical_accuracy: 0.2956 - val_loss: 2.1728 - val_sparse_categorical_accuracy: 0.2882
```

Outputsoftmax 50 Iterations

```python
#@title Outputsoftmax 50 Iterations
Model_exec(nbr_neural=20 ,batch_size=1000,epochs=50, output_Activation_function='softmax' , activation_function='sigmoid')
```

```
# Fit model on training data
Epoch 1/50
50/50 [==============================] - 0s 9ms/step - loss: 2.2851 - sparse_categorical_accuracy: 0.1726 - val_loss: 2.2768 - val_sparse_categorical_accuracy: 0.1838
Epoch 2/50
50/50 [==============================] - 0s 7ms/step - loss: 2.2694 - sparse_categorical_accuracy: 0.1902 - val_loss: 2.2721 - val_sparse_categorical_accuracy: 0.1886
Epoch 3/50
50/50 [==============================] - 0s 7ms/step - loss: 2.2687 - sparse_categorical_accuracy: 0.1915 - val_loss: 2.2692 - val_sparse_categorical_accuracy: 0.1874
```

…….

```
Epoch 48/50
50/50 [==============================] - 0s 7ms/step - loss: 2.1814 - sparse_categorical_accuracy: 0.2797 - val_loss: 2.1787 - val_sparse_categorical_accuracy: 0.2824
Epoch 49/50
50/50 [==============================] - 0s 7ms/step - loss: 2.1798 - sparse_categorical_accuracy: 0.2813 - val_loss: 2.1750 - val_sparse_categorical_accuracy: 0.2861
Epoch 50/50
50/50 [==============================] - 0s 7ms/step - loss: 2.1749 - sparse_categorical_accuracy: 0.2861 - val_loss: 2.1758 - val_sparse_categorical_accuracy: 0.2853
```

Auto-Encoder

```python
#@title Importation Pour AutoEncoder
from tensorflow.keras.layers import Dense, Input
from tensorflow.keras.layers import Conv2D, Flatten
from tensorflow.keras.layers import Reshape, Conv2DTranspose
from tensorflow.keras.models import Model
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import plot_model
from tensorflow.keras import backend as K
```

Importation Pour AutoEncoder

```
#@title Initialisation des données
# Charger le jeu de données MNIST
(x_train, y_train), (x_test, y_test) = mnist.load_data()
# reshape en (28, 28, 1)
image_size = x_train.shape[1]
x_train = np.reshape(x_train, [-1, image_size, image_size, 1
x_test = np.reshape(x_test, [-1, image_size, image_size, 1])
# normaliser
x_train = x_train.astype('float32') / 255
x_test = x_test.astype('float32') / 255
input_shape = (image_size, image_size, 1)
batch_size = 100
kernel_size = 3
latent_dim = 16
layer_filters = [32, 64]
inputs = Input(shape=input_shape, name='Encoder_input')
latent_inputs = Input(shape=(latent_dim,), name='Decoder_inp
```

Initialisation des données

```
#@title Encoder
def Encoder(inputs,input_shape,kernel_size,latent_dim,layer_
  inputs = Input(shape=input_shape, name='Encoder_input')
  x = inputs
  # Conv2D(32)-Conv2D(64)
  for filters in layer_filters:
      x = Conv2D(filters=filters,
                 kernel_size=kernel_size,
                 activation='relu',
                 strides=2,
                 padding='same')(x)
  shape = K.int_shape(x)

  # générer un vecteur latent
  x = Flatten()(x)
  latent = Dense(latent_dim, name='latent_vector')(x)

  encoder = Model(inputs, latent, name='Encoder')
  encoder.summary()
  plot_model(encoder, to_file='Encoder.png', show_shapes=Tru
  return encoder,shape
```

Encoder

```
#@title Decoder
def Decoder(latent_inputs,kernel_size,layer_filters,shape):
  x = Dense(shape[1] * shape[2] * shape[3])(latent_inputs)
  x = Reshape((shape[1], shape[2], shape[3]))(x)
  # Conv2DTranspose(64)-Conv2DTranspose(32)
  for filters in layer_filters[::-1]:
      x = Conv2DTranspose(filters=filters,
                          kernel_size=kernel_size,
                          activation='relu',
                          strides=2,
                          padding='same')(x)

  outputs = Conv2DTranspose(filters=1,
                            kernel_size=kernel_size,
                            activation='sigmoid',
                            padding='same',
                            name='Decoder_output')(x)

  # instantiate decoder model
  decoder = Model(latent_inputs, outputs, name='Decoder')
  decoder.summary()
  plot_model(decoder, to_file='Decoder.png', show_shapes=Tru
  return decoder
```

Decoder

```python
#@title Auto-Encoder = Encoder + Decoder
def AutoEncoder(inputs,batch_size,encoder,decoder,x_train,x_
  autoencoder = Model(inputs, decoder(encoder(inputs)), name
  autoencoder.summary()
  plot_model(autoencoder, to_file='Auto_Encoder.png', show_s
  autoencoder.compile(loss='mse', optimizer='adam')
  autoencoder.fit(x_train,
                  x_train,
                  validation_data=(x_test, x_test),
                  epochs=1,
                  batch_size=batch_size)
  x_decoded = autoencoder.predict(x_test)
  return autoencoder,x_decoded
```

Auto-Encoder = Encoder + Decoder

```python
#@title Run
encoder,shape = Encoder(inputs,input_shape,kernel_size,laten
decoder = Decoder(latent_inputs,kernel_size,layer_filters,sh
autoencoder,x_decoded = AutoEncoder(inputs,batch_size,encode
```

Run

```
Model: "Encoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Encoder_input (InputLayer)   [(None, 28, 28, 1)]       0
_________________________________________________________________
conv2d (Conv2D)              (None, 14, 14, 32)        320
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 7, 7, 64)          18496
_________________________________________________________________
flatten (Flatten)            (None, 3136)              0
_________________________________________________________________
latent_vector (Dense)        (None, 16)                50192
=================================================================
Total params: 69,008
Trainable params: 69,008
Non-trainable params: 0
_________________________________________________________________
Model: "Decoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Decoder_input (InputLayer)   [(None, 16)]              0
_________________________________________________________________
dense (Dense)                (None, 3136)              53312
_________________________________________________________________
reshape (Reshape)            (None, 7, 7, 64)          0
_________________________________________________________________
conv2d_transpose (Conv2DTran (None, 14, 14, 64)        36928
_________________________________________________________________
conv2d_transpose_1 (Conv2DTr (None, 28, 28, 32)        18464
_________________________________________________________________
Decoder_output (Conv2DTransp (None, 28, 28, 1)         289
=================================================================
Total params: 108,993
Trainable params: 108,993
Non-trainable params: 0
_________________________________________________________________
Model: "Autoencoder"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Encoder_input (InputLayer)   [(None, 28, 28, 1)]       0
_________________________________________________________________
Encoder (Model)              (None, 16)                69008
_________________________________________________________________
Decoder (Model)              (None, 28, 28, 1)         108993
=================================================================
Total params: 178,001
Trainable params: 178,001
Non-trainable params: 0
_________________________________________________________________
600/600 [==============================] - 89s 149ms/step - loss: 0.0372 - val_loss: 0.0134
```

```
#@title Image final
imgs = np.concatenate([x_test[:8], x_decoded[:8]])
imgs = imgs.reshape((4, 4, image_size, image_size))
imgs = np.vstack([np.hstack(i) for i in imgs])
plt.figure()
plt.imshow(imgs, interpolation='none', cmap='gray')
plt.savefig('input_and_decoded.png')
plt.show()
```

Image final