Artificial Intelligence

Artificial Neural Networks
Hands-on

Slides by Carlos Grilo & Rolando Miragaia

An MLP network for the XOR function

An MLP network for the XOR function

- In our first exercise we will train an MLP neural network to work as an XOR logic function
- For this, we will create a copy of the Artificial Neurons hands-on notebook from previous classes and then modify some important details
- Copy notebook

    `02_artificial_neurons_01_AND.ipynb`

  and rename it as

    `03_MLP_01_XOR.ipynb`

An MLP network for the XOR function

- Replace code cell

```
# Defining the dataset
training_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], "float32")
target_data = np.array([[0], [0], [0], [1]], "float32")
```

for this new one

```
# Defining the dataset
training_data = np.array([[0, 0], [0, 1], [1, 0], [1, 1]], "float32")
target_data = np.array([[0], [1], [1], [0]], "float32")
```

An MLP network for the XOR function

- Replace code cell

```
# Defining the model
model = Sequential()
model.add(Dense(1, input_dim=2, activation='sigmoid'))
```

for this new one

```
# Defining the model
model = Sequential()
model.add(Dense(2, input_dim=2, activation='sigmoid'))
model.add(Dense(1, activation='sigmoid'))
```

Here, we are defining a NN with a hidden layer with two units and an output layer with one unit

An MLP network for the XOR function

- Now, try to adjust the hyper-parameters of the learning process in order to train the neural network so that it can model the XOR function

- After a successful training process, you can see the weights using

```
model.weights
```

```
[<tf.Variable 'dense/kernel:0' shape=(2, 2) dtype=float32, numpy=
array([[-5.728402 , 4.017226 ],
       [-5.7106543, 4.0139284]], dtype=float32)>,
<tf.Variable 'dense/bias:0' shape=(2,) dtype=float32, numpy=array([ 2.0943527, -6.243998 ],
dtype=float32)>,
<tf.Variable 'dense_1/kernel:0' shape=(2, 1) dtype=float32, numpy=
array([[-7.740917 ],
       [-7.8598757]], dtype=float32)>,
<tf.Variable 'dense_1/bias:0' shape=(1,) dtype=float32, numpy=array([3.843351], dtype=float32)>]
```

Recognizing handwritten digits

The MNIST dataset

- We will now build an MLP Network and train it with the MNIST dataset
- The MNIST dataset is a dataset of handwritten digits that is commonly used for training various image processing systems
- It contains 60000 training images and 10000 testing images
- It comes preloaded in Keras, in the form of a set of four Numpy arrays

Creating the notebook

- Create a new notebook and name it `03_MLP_02_MNIST.ipynb`

Loading the dataset

```
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

- train_images and train_labels form the *training set*, the data that the model will learn from
- The model will be tested on the *test set*, which is composed of test_images and test_labels
- Images are greyscale images encoded as numpy 2D arrays
- Pixel values represent the color intensity and vary between 0, for white, and 255, for black
- Labels are an array of digits, ranging from 0 to 9
- Images and labels have a one-to-one correspondence

Looking at the training data - images

```
>>> train_images.ndim
    3

>>> train_images.shape
    (60000, 28, 28)

>>> print(train_images.dtype)
    uint8
```

→ 60000 images of 28 x 28 pixels

- train_images is a 3D tensor (array) of 8-bit unsigned integers
- Each integer represents a pixel intensity (integer value between 0 and 255)

Looking at the training data - labels

```
>>> len(train_labels)
    60000
```

60000 labels

```
>>> train_labels
    array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

- `dtype=uint8` means that the labels are 8-bit unsigned integers
- However, the actual values range from 0 to 9, corresponding to the 10 digits

Looking at the test data

```
>>> test_images.ndim
    3

>>> test_images.shape
    (10000, 28, 28)
```

→ 10000 images of 28 x 28 pixels

```
>>> len(test_labels)
    10000
```

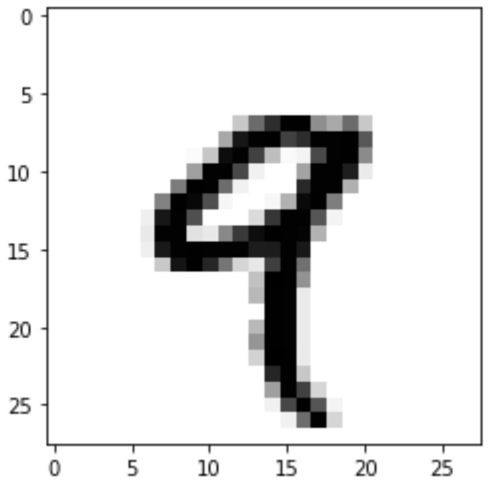
→ 10000 labels

```
>>> test_labels
    array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

Example of a training image

```
digit = train_images[4]

import matplotlib.pyplot as plt
plt.imshow(digit, cmap=plt.cm.binary)
plt.show()
```



Printing pixel intensity values

```python
for row in digit:
    for elem in row:
        print("%3d" % (elem), end=" ")
    print()
```

```
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0  55 148 210 253 253 113  87 148  55   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0  87 232 252 253 189 210 252 252 253 168   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   4  57 242 252 190  65   5  12 182 252 253 116   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0  96 252 252 183  14   0   0  92 252 252 225  21   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0 132 253 252 146  14   0   0   0 215 252 252  79   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0 126 253 247 176   9   0   0   8  78 245 253 129   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0  16 232 252 176   0   0   0  36 201 252 252 169  11   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0  22 252 252  30  22 119 197 241 253 252 251  77   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0  16 231 252 253 252 252 252 226 227 252 231   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0  55 235 253 217 138  42  24 192 252 143   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0  62 255 253 109   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0  71 253 252  21   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0 253 252  21   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0  71 253 252  21   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0 106 253 252  21   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0  45 255 253  21   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0 218 252  56   0   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0  96 252 189  42   0   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0  14 184 252 170  11   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0  14 147 252  42   0   0   0   0   0   0   0   0   0
  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Building the model

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))
```

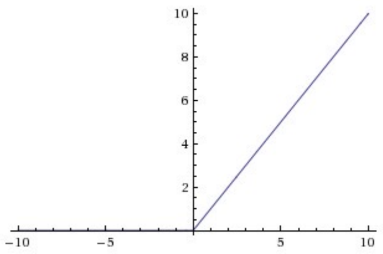
- The network consists of a sequence of two Dense layers

Building the model

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))
```

- The first layer has 512 units
- It is a ReLU layer
- It has 28 x 28 inputs - the number of pixels of each image (784)



ReLU

Building the model

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))
```

- The second (and last) layer is a 10-way softmax layer $\longrightarrow f(z)_i = \frac{e^{z_i}}{\sum_{i=1}^{K} e^{z_i}}$
- It will return an array of 10 probability scores (summing to 1)
- Each score is the probability that the current digit image belongs to one of the 10 digit classes

Compiling the model

```
model.compile(optimizer='rmsprop',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

- The default learning rate for the rmsprop optimizer is equal to 0.001
- categorical_crossentropy should be used for multiclass, single-label classification problems
- accuracy: the fraction of the images that were correctly classified

Reshaping the training and test data

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

- Previously, the training images were stored in an array of shape (60000, 28, 28) of type uint8 with values in the [0, 255] interval
- We now transform it into a float32 array of shape (60000, 28 * 28) with values between 0 and 1

Reshaping the training and test data

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

- Why an array of shape (60000, 28 * 28)? Because the network receives an array of 28 x 28 values as input per image (60000 is the number of images)

- Why a float32 array with values between 0 and 1?
  Because the Backpropagation algorithm works better if input values are small (e.g. in the [0, 1] or [-1, 1] intervals)

Preparing the labels

```
from keras.utils import to_categorical

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

- We need to convert the labels to a categorical representation
- For example:
  - label 0 is transformed into [1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
  - label 1 is transformed into [0, 1, 0, 0, 0, 0, 0, 0, 0, 0]
  - and so on

```
print(train_labels.shape)
print(train_labels[0])

(60000, 10)
[0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

Training the model

```
>>> model.fit(train_images, train_labels, epochs=5, batch_size=128)
    Epoch 1/5
    469/469 [==============================] - 4s 7ms/step - loss: 0.4267 - accuracy: 0.8757
    Epoch 2/5
    469/469 [==============================] - 3s 7ms/step - loss: 0.1127 - accuracy: 0.9669
    Epoch 3/5
    469/469 [==============================] - 3s 7ms/step - loss: 0.0676 - accuracy: 0.9806
    Epoch 4/5
    469/469 [==============================] - 3s 7ms/step - loss: 0.0500 - accuracy: 0.9855
    Epoch 5/5
    469/469 [==============================] - 3s 7ms/step - loss: 0.0355 - accuracy: 0.9898
```

Assessing the model's performance

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

```
313/313 [==========================] - 1s 2ms/step - loss: 0.0648 - accuracy: 0.9812
test_acc: 0.9811999797821045
```

- Save the model

```
model.save("models/03_MLP_02_MNIST.h5")
```

VSCode

Assessing the model's performance

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

```
313/313 [========================] - 1s 2ms/step - loss: 0.0648 - accuracy: 0.9812
test_acc: 0.9811999797821045
```

- Save the model

```
from google.colab import drive
drive.mount('/content/drive')
-------------

model.save("/content/drive/MyDrive/models/03_MLP_02_MNIST.h5")
```

Colab

Exercises

- Try different variations of what we have done. For example, you can:
  - try different values for the learning rate
  - try different values for the batch size
  - use alternative activation functions
  - use the SGD optimizer
  - vary the number of units of the hidden layer
  - use 3 layers (or more) and vary the number of units of the hidden layers