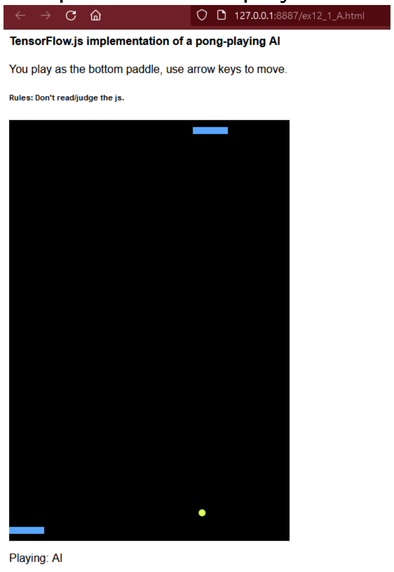
物聯網實務第十三周作業

電機四乙10828241 陳大荃

December 7, 2022

Exercise 12-1 Train model in python.

This is the result following the instruction of `https://pythonprogramming.net/loading-keras-model-te`
According to the JavaScript file provided on the website, it loads a pre-trained model and uses it to compete with the player.



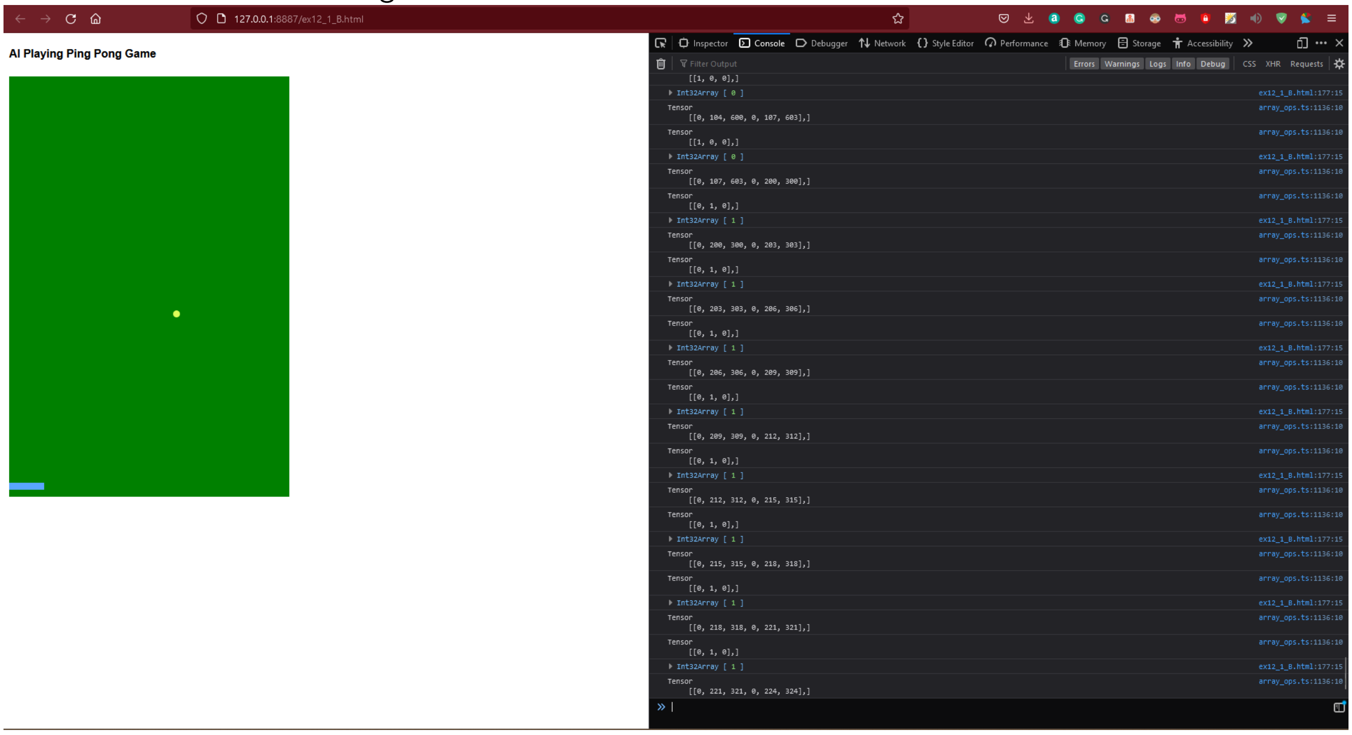
TensorFlow.js implementation of a pong-playing AI

You play as the bottom paddle, use arrow keys to move.

Rules: Don't read/judge the js.

Playing: AI

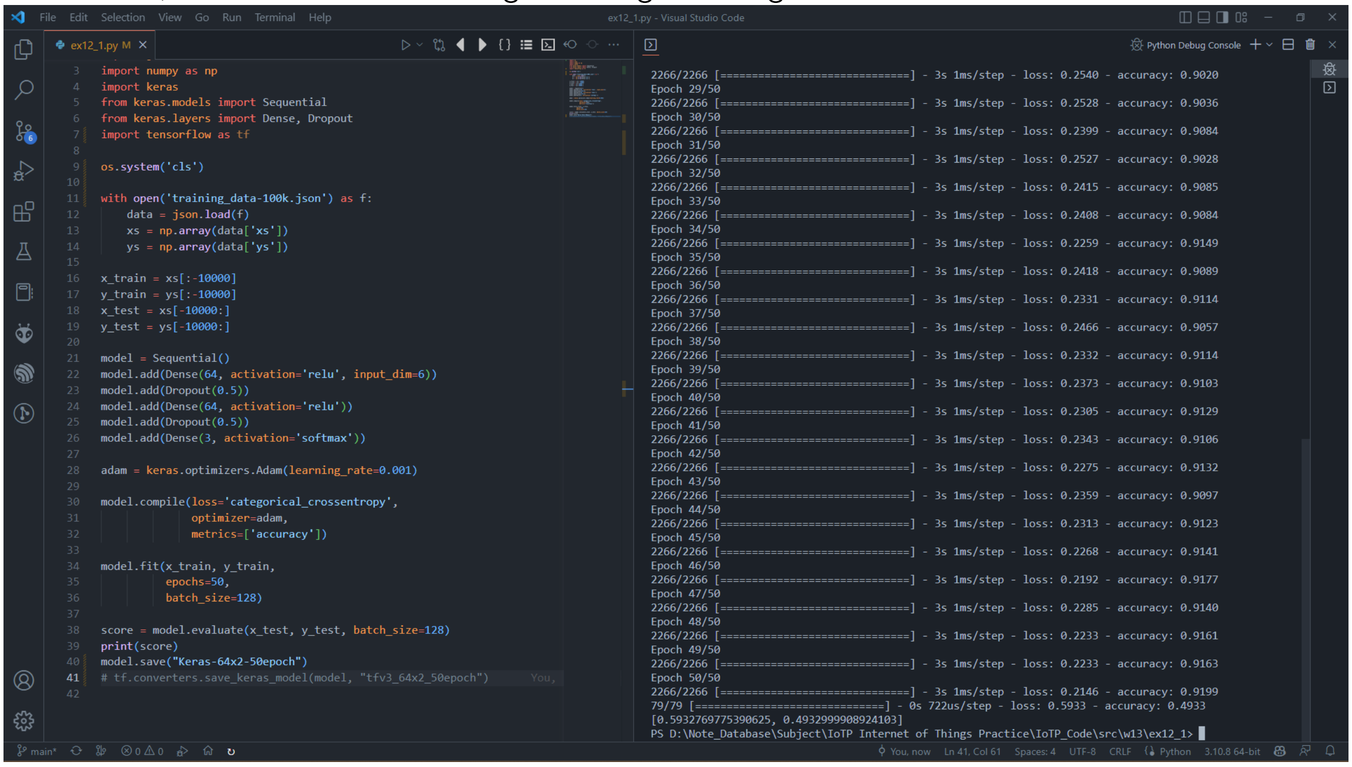
This is the result following the instruction in class material.



AI Playing Ping Pong Game

```
    [[1, 0, 0],]
▸ Int32Array [ 0 ]                                  ex12_1_B.html:177:15
Tensor                                              array_ops.ts:1136:10
    [[0, 104, 600, 0, 107, 603],]
Tensor                                              array_ops.ts:1136:10
    [[1, 0, 0],]
▸ Int32Array [ 0 ]                                  ex12_1_B.html:177:15
Tensor                                              array_ops.ts:1136:10
    [[0, 107, 603, 0, 200, 300],]
Tensor                                              array_ops.ts:1136:10
    [[0, 1, 0],]
▸ Int32Array [ 1 ]                                  ex12_1_B.html:177:15
Tensor                                              array_ops.ts:1136:10
    [[0, 200, 300, 0, 203, 303],]
Tensor                                              array_ops.ts:1136:10
    [[0, 1, 0],]
▸ Int32Array [ 1 ]                                  ex12_1_B.html:177:15
Tensor                                              array_ops.ts:1136:10
    [[0, 203, 303, 0, 206, 306],]
Tensor                                              array_ops.ts:1136:10
    [[0, 1, 0],]
▸ Int32Array [ 1 ]                                  ex12_1_B.html:177:15
Tensor                                              array_ops.ts:1136:10
    [[0, 206, 306, 0, 209, 309],]
Tensor                                              array_ops.ts:1136:10
    [[0, 1, 0],]
▸ Int32Array [ 1 ]                                  ex12_1_B.html:177:15
Tensor                                              array_ops.ts:1136:10
    [[0, 209, 309, 0, 212, 312],]
Tensor                                              array_ops.ts:1136:10
    [[0, 1, 0],]
▸ Int32Array [ 1 ]                                  ex12_1_B.html:177:15
Tensor                                              array_ops.ts:1136:10
    [[0, 212, 312, 0, 215, 315],]
Tensor                                              array_ops.ts:1136:10
    [[0, 1, 0],]
▸ Int32Array [ 1 ]                                  ex12_1_B.html:177:15
Tensor                                              array_ops.ts:1136:10
    [[0, 215, 315, 0, 218, 318],]
Tensor                                              array_ops.ts:1136:10
    [[0, 1, 0],]
▸ Int32Array [ 1 ]                                  ex12_1_B.html:177:15
Tensor                                              array_ops.ts:1136:10
    [[0, 218, 318, 0, 221, 321],]
Tensor                                              array_ops.ts:1136:10
    [[0, 1, 0],]
▸ Int32Array [ 1 ]                                  ex12_1_B.html:177:15
Tensor                                              array_ops.ts:1136:10
    [[0, 221, 321, 0, 224, 324],]
»
```

It is also possible to train with a local machine with "tensorflow" instead of "tensorflowjs." Using this method, we can utilize a much larger training set and get a decent result.

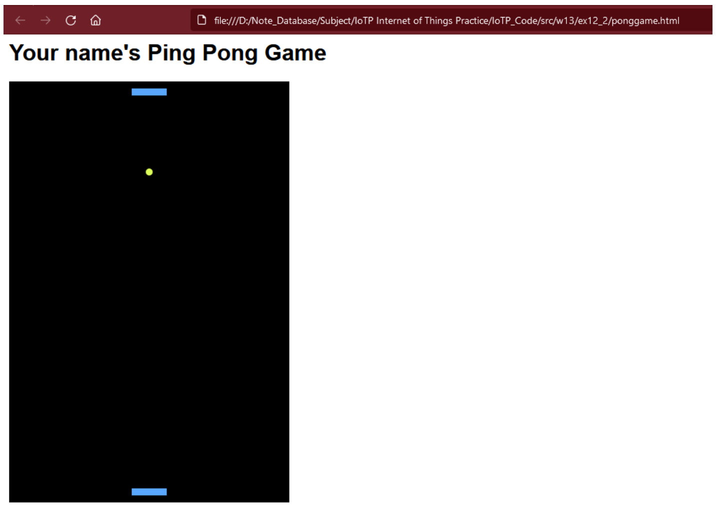

File Edit Selection View Go Run Terminal Help
ex12_1.py - Visual Studio Code
ex12_1.py M
Python Debug Console
main*
You, now Ln 41, Col 61 Spaces: 4 UTF-8 CRLF Python 3.10.8 64-bit

```python
import numpy as np
import keras
from keras.models import Sequential
from keras.layers import Dense, Dropout
import tensorflow as tf

os.system('cls')

with open('training_data-100k.json') as f:
    data = json.load(f)
    xs = np.array(data['xs'])
    ys = np.array(data['ys'])

x_train = xs[:-10000]
y_train = ys[:-10000]
x_test = xs[-10000:]
y_test = ys[-10000:]

model = Sequential()
model.add(Dense(64, activation='relu', input_dim=6))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(3, activation='softmax'))

adam = keras.optimizers.Adam(learning_rate=0.001)

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

model.fit(x_train, y_train,
          epochs=50,
          batch_size=128)

score = model.evaluate(x_test, y_test, batch_size=128)
print(score)
model.save("Keras-64x2-50epoch")
# tf.converters.save_keras_model(model, "tfv3_64x2_50epoch")
```

```
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2540 - accuracy: 0.9020
Epoch 29/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2528 - accuracy: 0.9036
Epoch 30/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2399 - accuracy: 0.9084
Epoch 31/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2527 - accuracy: 0.9028
Epoch 32/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2415 - accuracy: 0.9085
Epoch 33/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2408 - accuracy: 0.9084
Epoch 34/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2259 - accuracy: 0.9149
Epoch 35/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2418 - accuracy: 0.9089
Epoch 36/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2331 - accuracy: 0.9114
Epoch 37/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2466 - accuracy: 0.9057
Epoch 38/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2332 - accuracy: 0.9114
Epoch 39/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2373 - accuracy: 0.9103
Epoch 40/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2305 - accuracy: 0.9129
Epoch 41/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2343 - accuracy: 0.9106
Epoch 42/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2275 - accuracy: 0.9132
Epoch 43/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2359 - accuracy: 0.9097
Epoch 44/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2313 - accuracy: 0.9123
Epoch 45/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2268 - accuracy: 0.9141
Epoch 46/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2192 - accuracy: 0.9177
Epoch 47/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2285 - accuracy: 0.9140
Epoch 48/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2233 - accuracy: 0.9161
Epoch 49/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2233 - accuracy: 0.9163
Epoch 50/50
2266/2266 [==============================] - 3s 1ms/step - loss: 0.2146 - accuracy: 0.9199
79/79 [==============================] - 0s 722us/step - loss: 0.5933 - accuracy: 0.4933
[0.5932769775390625, 0.4932999908924103]
PS D:\Note_Database\Subject\IoTP Internet of Things Practice\IoTP_Code\src\w13\ex12_1>
```

Exercise 12-2 Pong clone in JavaScript.

I uploaded the details about how this code runs at `https://github.com/belongtothenight/IoTP_Code/blob/main/src/w12/ex11_7.md`. Basically, it utilizes the data player created and trains new models according to it in real-time.


file:///D:/Note_Database/Subject/IoTP Internet of Things Practice/IoTP_Code/src/w13/ex12_2/ponggame.html

Your name's Ping Pong Game