The Impossible Game A.I.

Javier Holgueras Crespo
Pedro Burgos Gonzalo
Sergio García Bustos

Problema a resolver

¿Qué Problema?

- Cualidades que buscamos en el problema
  - Datos accesibles
  - Error bien definido
  - Entradas bien definidas y abarcables
  - Salidas bien definidas y abarcables

¿Qué Problema?

- ¿Por qué un juego?
  - Podemos generar datos nosotros mismos
  - Tienen una finalidad claramente definida -> Ganar
  - Entradas bien definidas -> La pantalla
  - Salida bien definidas -> Los controles

¿Qué Problema?

- Cualidades que buscamos en el juego
  - El ganar o perder una partida debe estar bien definido
  - Los controles no deben de ser demasiados o demasiado complicados
  - No debe ser muy pesado de ejecutar
  - Debemos poder conseguirlo fácilmente

¿Qué Problema?

- ¿Por qué The Impossible Game?
  - No es muy costoso de ejecutar
  - Es un juego con reglas sencillas
  - Tiene controles sencillos
  - Se puede obtener fácilmente
  - No requiere de toda la pantalla para jugar
  - No requiere de una alta resolución para jugar

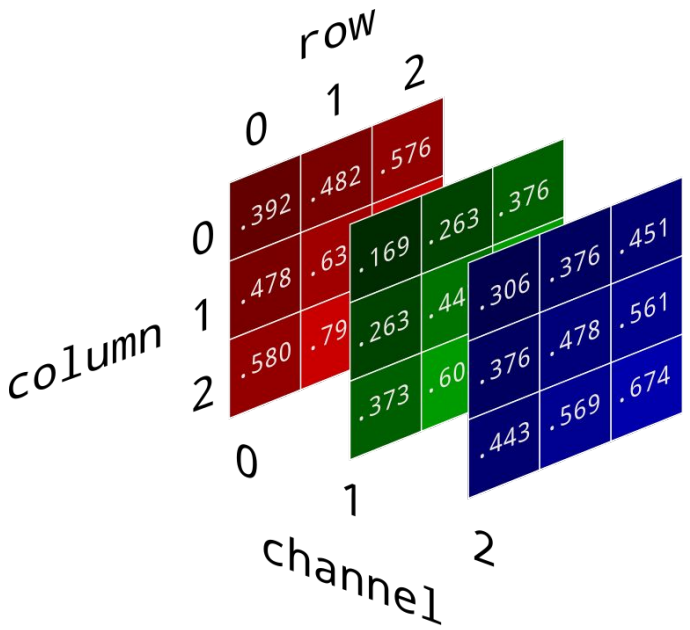
Captura y tratamiento de los datos

¿Cómo tomamos los datos del juego?

- Datos para entrenamiento / Datos durante la ejecución (live)
- ¿Que opciones tenemos?
  - Programar una copia del juego nosotros
  - Obtener posiciones de los objetos inspeccionando la memoria
  - Grabar la pantalla

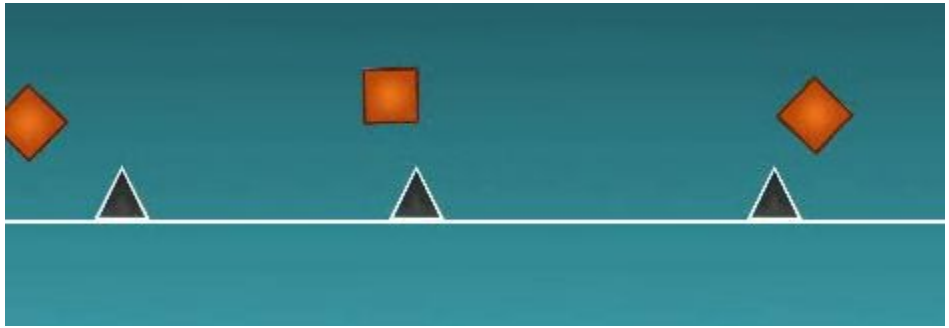
Datos: Captura

- Video: Secuencias de imágenes
- Cada frame es un array Numpy con los valores RGB de cada pixel
- Cargamos frame por frame mientras jugamos



Datos: Secuencialidad

- ¿Cómo sabemos en qué dirección se realiza el movimiento?
    - Solución: Agrupar varias imágenes en cada dato del dataset



Datos: Velocidad

- Procesar imágenes es lento, y necesitamos procesarlas en tiempo real
  - Reducir la resolución
  - Convertir a escala de grises (?)

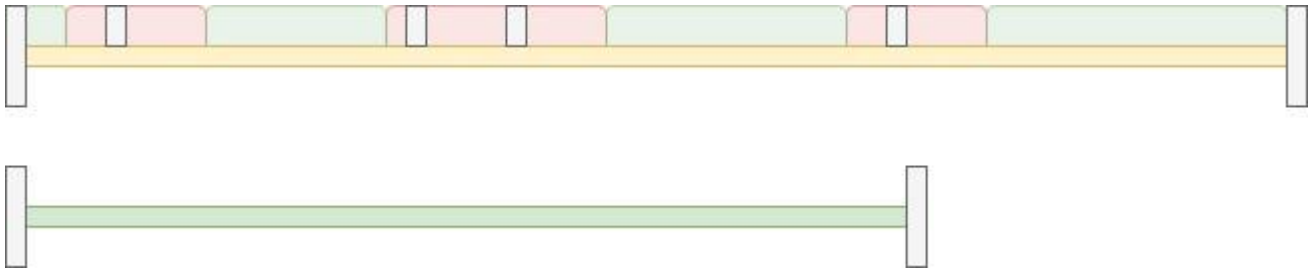
Datos: Captura de teclas

- Hasta ahora solo hemos hablado de datos, ¿Cual es la clase?
  - La clase es “PULSAR ESPACIO” o “NO PULSAR ESPACIO”
- ¿Cómo etiquetamos los datos?
  - Para cada frame capturado, registramos las pulsaciones de las teclas que usemos

¿Cómo conseguimos generar un dataset correcto?

- Somos muy malos
  - Si la red aprende de nosotros, va a aprender a morir

- Solución: Detectamos las muertes y las retiramos del video



Al final...

- Al final del procesado tenemos:
  - dataset.shape(N_DATOS)
  - dato.shape([frame1, frame2, frame3,], KEY)

Modelo

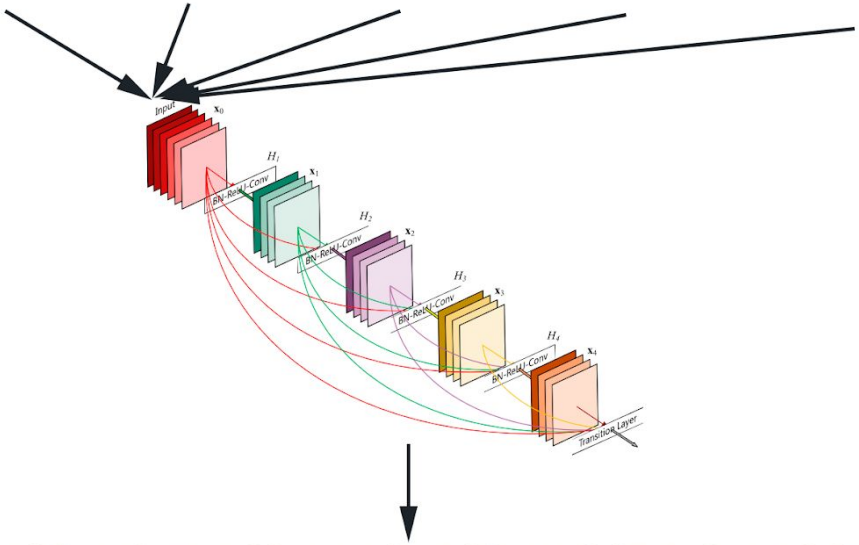
Fuente del modelo

- https://github.com/ikergarcia1996/Self-Driving-Car-in-Video-Games
- Construida utilizando el framework PyTorch
- 3 capas de redes diferentes:
  - Red neuronal convolucional profunda (Deep Convolutional Neural Network)
    - Residual neural network (Resnet)
  - Red neuronal recurrente (Recurrent Neural Network o LSTM)
  - Red neuronal prealimentada (Feed-Forward Neural Network)
    - Perceptrón multicapa

Modelo base

- Red Neuronal Convolucional Profunda (Deep Convolutional Neural Network)
  - Construida utilizando un tipo especial de red llamada Resnet (Residual Neural Network)
  - Las redes Resnet tiene la características de poder saltar capas mediante atajos.
  - Esta red se encarga de procesar las imágenes para extraer las características.
  - Está preentrenada.
  - Recibe 5 imágenes de entrada y las procesa a un único vector por imagen

Modelo base



RESNET

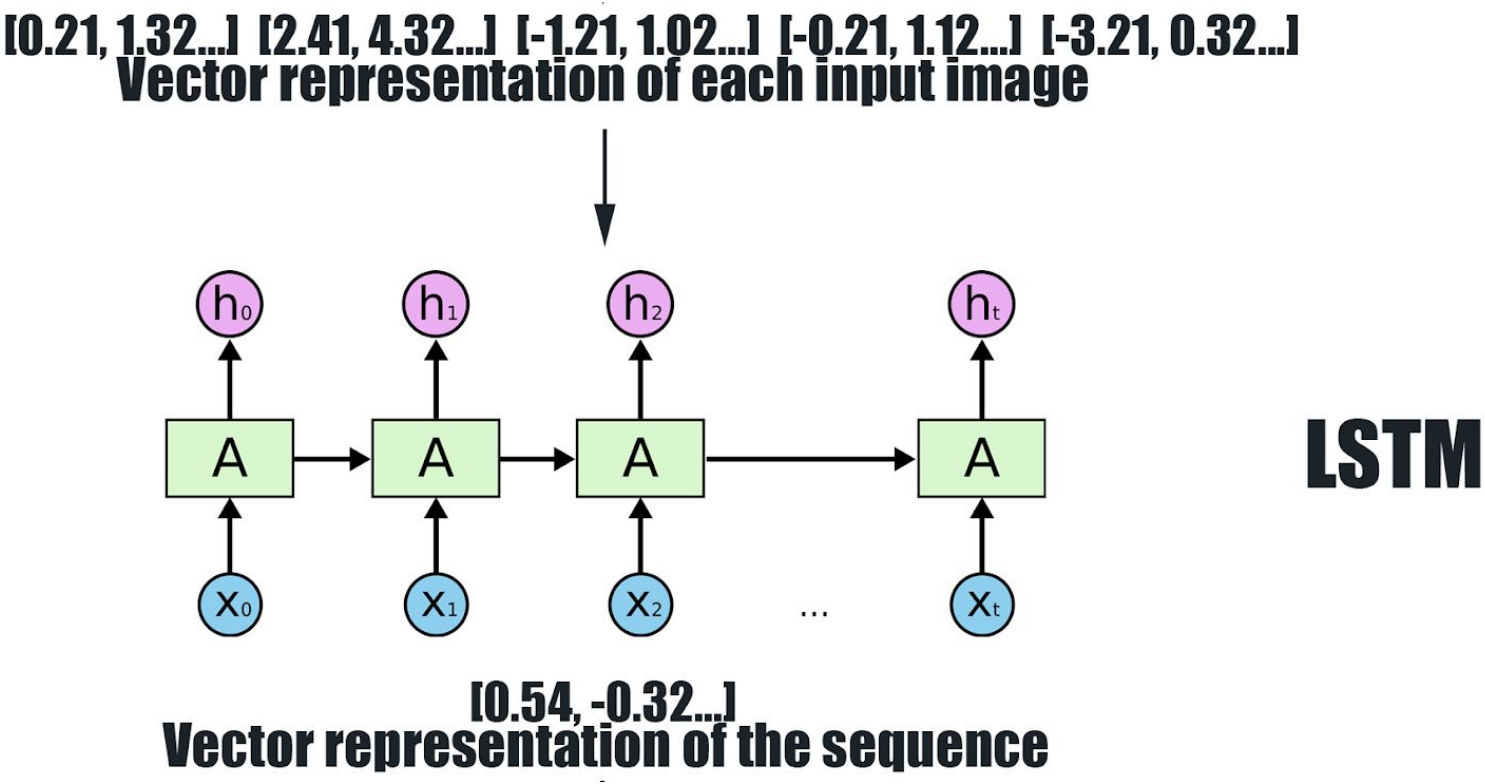
[0.21, 1.32...] [2.41, 4.32...] [-1.21, 1.02...] [-0.21, 1.12...] [-3.21, 0.32...]

Vector representation of each input image

Modelo base

- Red Neuronal Recurrente (Recurrent Neural Network o LSTM)
  - Se puede realimentar a sí mismas
  - Guardan información durante algunos pasos o épocas.

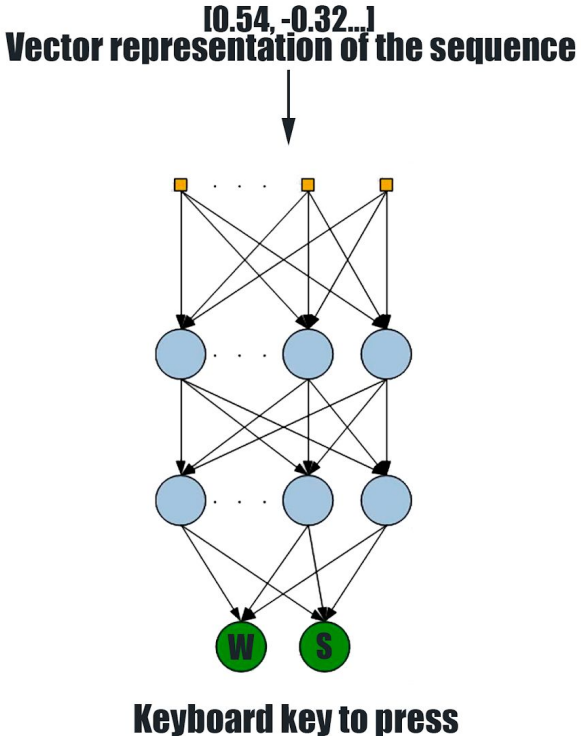
Modelo base


[0.21, 1.32...] [2.41, 4.32...] [-1.21, 1.02...] [-0.21, 1.12...] [-3.21, 0.32...]
Vector representation of each input image
h0
h1
h2
ht
A
A
A
A
x0
x1
x2
...
xt
LSTM
[0.54, -0.32...]
Vector representation of the sequence

Modelo base

- Red neuronal prealimentada (Feed-Forward Neural Network)
  - Implementada mediante un perceptrón multicapa
  - Toma las decisiones

Modelo base



FNN

Problemas del modelo base

- Velocidad de procesado
- Complejidad de la red
- Framework que no conocemos

Objetivos de nuestro modelo

- Implementado mediante el framework Keras de Tensorflow, que conocemos mejor
- Más simple
- Más rápido

Nuestro modelo

```
model = Sequential()

model.add(InputLayer(input_shape=(num_img_per_seq, img_res[0], img_res[1], 1)))

model.add(TimeDistributed(Convolution2D(32, (4, 4), activation='relu')))
model.add(TimeDistributed(MaxPooling2D(pool_size=(5, 5))))
model.add(TimeDistributed(Convolution2D(16, (4, 4), activation='relu')))
model.add(TimeDistributed(MaxPooling2D(pool_size=(5, 5))))
model.add(TimeDistributed(Dropout(0.25)))
model.add(TimeDistributed(Flatten()))

model.add(GRU(128, kernel_initializer=initializers.RandomNormal(stddev=0.001)))  # 128
model.add(Dropout(0.25))

model.add(Dense(60))
model.add(Dense(40))
model.add(Dense(num_cls, activation='sigmoid'))

opt = optimizers.RMSprop(lr=0.001)
model.compile(loss='mean_squared_error', optimizer=opt, metrics=['accuracy'])
```

Entrenamiento

- Train: 3 ficheros con 6 min de juego
- Eval: 1 ficheros con 4 min de juego
- Test: 1 ficheros con 4 min de juego
- 5 épocas
- Tiempo de entrenamiento: 10 min
- Precisión en Test: ~0.85

Resultados

- 3 veces más rápido en la toma de decisiones
- 20 decisiones por segundo

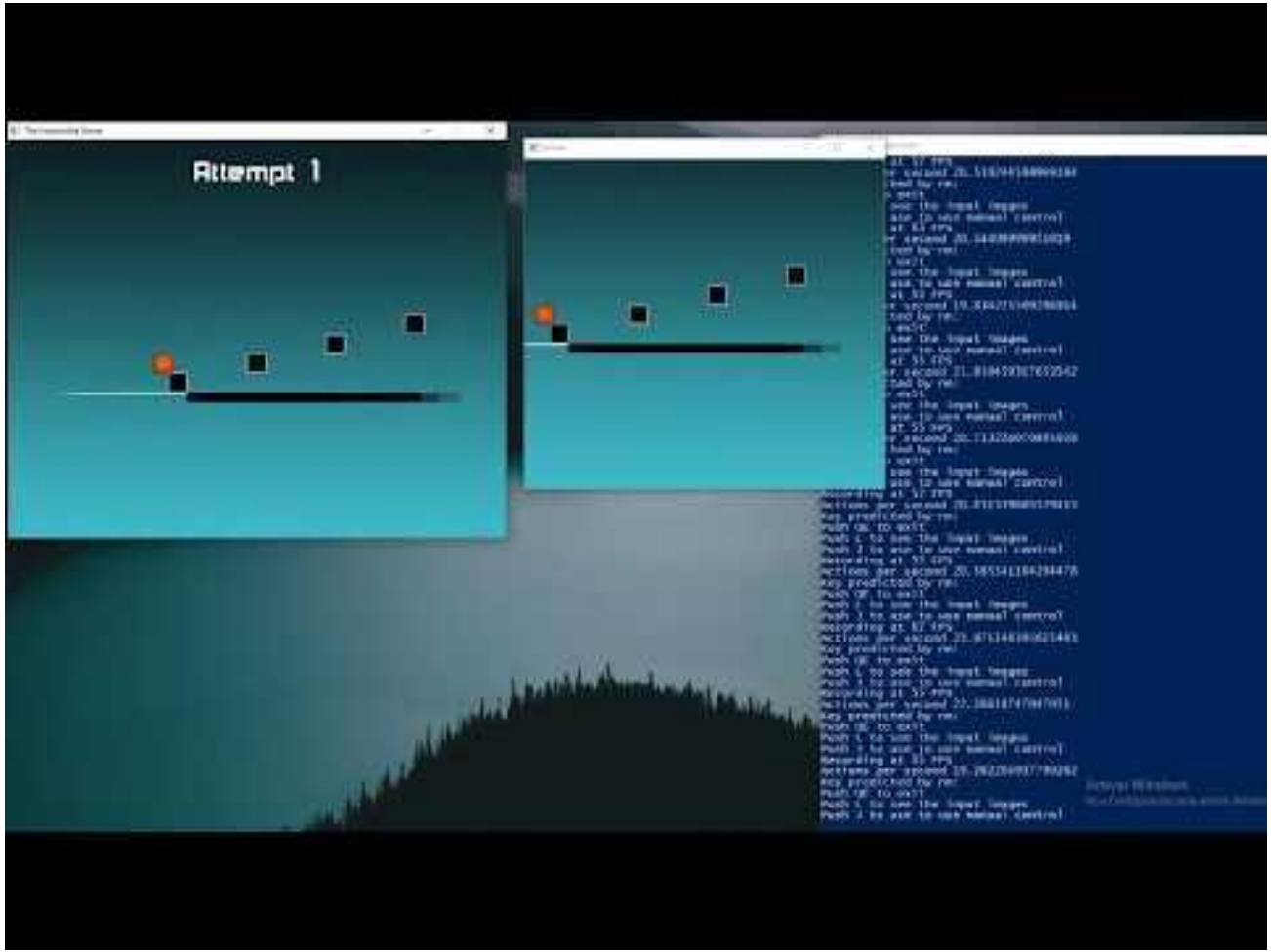
Problemas

- Todavía es demasiado lento en la toma de decisiones
- 60 fps de grabación

Solución

- Ejecutar la red neuronal en un ordenador con mayor capacidad de cómputo para redes neuronales.
- Reducir la velocidad del juego al 0.25 mediante CheatEngine

Vídeo



Referencias

- T.E.D.D. 1104: https://github.com/ikergarcia1996/Self-Driving-Car-in-Video-Games
- Grabber: https://gist.github.com/tzickel/5c2c51ddde7a8f5d87be730046612cd0
- Github del proyecto: https://github.com/druskus20/impossible_game_ai