Tarea 5

Matemáticas para las ciencias de la computación

Dos ejercicios de cada libro:

- **Artificial Intelligence with Python**
- **Artificial Intelligence by Example**
- **Deep learning illustrated**

Por: Miguel Angel Soto Hernandez

Importaciones e instalaciones necesarias

Instalaciones

In [1]:

```
!pip install easyAI
```

```
Collecting easyAI
  Downloading https://files.pythonhosted.org/packages/54/a0/c81556cd42db23a545a330e9331f9
6fd658fb4098392f7f686fa5c9b8836/easyAI-1.0.0.4.tar.gz
Building wheels for collected packages: easyAI
  Building wheel for easyAI (setup.py) ... done
  Created wheel for easyAI: filename=easyAI-1.0.0.4-py2.py3-none-any.whl size=41950 sha25
6=b169202049245b4a060a3be80b9370a80b4f605eacb4591008d3cb0d8568753b
  Stored in directory: /root/.cache/pip/wheels/0d/4c/ec/89fb6bed5865d245eca16ca7dce1aaa46
b631998a9c545163e
Successfully built easyAI
Installing collected packages: easyAI
Successfully installed easyAI-1.0.0.4
```

Importaciones

In [14]:

```
from easyAI import TwoPlayersGame, AI_Player, Negamax, Human_Player, SSS
from easyAI.Player import Human_Player
import numpy as np
import pandas as pd
import tensorflow as tf
from keras import Sequential
from keras.layers.normalization import BatchNormalization
from keras.layers import Dense, Activation, Dropout
from keras.initializers import Zeros, RandomNormal, glorot_normal, glorot_uniform
from keras.datasets import boston_housing
from tensorflow.keras import datasets, layers, models
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

Artificial Intelligence with Python

Ejercicio 1: Gato

In [ ]:

```
class GameController(TwoPlayersGame):
    def __init__(self, players):
        # Definiendo los jugadores
        self.players = players

        # Definiendo quien empezara el juego
        self.nplayer = 1

        # Definiendo el tablero
        self.board = [0] * 9


    # Definiendo los posibles movimientos
    def possible_moves(self):
        return [a + 1 for a, b in enumerate(self.board) if b == 0]


    # Hacer un movimiento
    def make_move(self, move):
        self.board[int(move) - 1] = self.nplayer


    # El oponente tiene 3 en linea?
    def loss_condition(self):
        possible_combinations = [[1,2,3], [4,5,6], [7,8,9],
            [1,4,7], [2,5,8], [3,6,9], [1,5,9], [3,5,7]]
        return any([all([(self.board[i-1] == self.nopponent)
                for i in combination]) for combination in possible_combinations])


    # Checar si el juego ya ha terminado
    def is_over(self):
        return (self.possible_moves() == []) or self.loss_condition()


    # Ver la posicion actual
    def show(self):
        print('\n'+'\n'.join([' '.join([['.', 'O', 'X'][self.board[3*j+i]]
                for i in range(3)]) for j in range(3)]))


    # Computar el score
    def scoring(self):
        return -100 if self.loss_condition() else 0


if __name__ == "__main__":
    # Definir el algoritmo
    algorithm = Negamax(7)

    # Empezar el juego
    GameController([Human_Player(), AI_Player(algorithm)]).play()
```

```
. . .
. . .
. . .

Player 1 what do you play ? 5

Move #1: player 1 plays 5 :

. . .
. O .
. . .

Move #2: player 2 plays 1 :

X . .
```

```
. O .
. . .

Player 1 what do you play ? 9

Move #3: player 1 plays 9 :

X . .
. O .
. . O

Move #4: player 2 plays 3 :

X . X
. O .
. . O

Player 1 what do you play ? 2

Move #5: player 1 plays 2 :

X O X
. O .
. . O

Move #6: player 2 plays 8 :

X O X
. O .
. X O

Player 1 what do you play ? 4

Move #7: player 1 plays 4 :

X O X
O O .
. X O

Move #8: player 2 plays 6 :

X O X
O O X
. X O

Player 1 what do you play ? 7

Move #9: player 1 plays 7 :

X O X
O O X
O X O
```

Ejercicio 2: Conecta Cuatro

In [ ]:

```
# Librerias necesarias
import numpy as np
from easyAI import TwoPlayersGame, Human_Player, AI_Player, \
        Negamax, SSS

# La clase GameController tendra todas las funciones necesarias para jugar
class GameController(TwoPlayersGame):
    def __init__(self, players, board = None):
        # Definiendo los jugadores
        self.players = players

        # Defininendo la configuracion del tablero
        self.board = board if (board != None) else (
```

```
            np.array([[0 for i in range(7)] for j in range(6)]))

        # Definiendo quien empezara el juego
        self.nplayer = 1

        # Definiendo las posiciones
        self.pos_dir = np.array([[[i, 0], [0, 1]] for i in range(6)] +
                 [[[0, i], [1, 0]] for i in range(7)] +
                 [[[i, 0], [1, 1]] for i in range(1, 3)] +
                 [[[0, i], [1, 1]] for i in range(4)] +
                 [[[i, 6], [1, -1]] for i in range(1, 3)] +
                 [[[0, i], [1, -1]] for i in range(3, 7)])


    # Definiendo los posibles movimientos
    def possible_moves(self):
        return [i for i in range(7) if (self.board[:, i].min() == 0)]


    # Definiendo como hacer un movimiento
    def make_move(self, column):
        line = np.argmin(self.board[:, column] != 0)
        self.board[line, column] = self.nplayer


    # Mostrar el estado actual
    def show(self):
        print('\n' + '\n'.join(
                ['0 1 2 3 4 5 6', 13 * '-'] +
                [' '.join([['.', 'O', 'X'][self.board[5 - j][i]]
                for i in range(7)]) for j in range(6)]))


    # Definiendo como se ve la condicion de perdida
    def loss_condition(self):
        for pos, direction in self.pos_dir:
            streak = 0
            while (0 <= pos[0] <= 5) and (0 <= pos[1] <= 6):
                if self.board[pos[0], pos[1]] == self.nopponent:
                    streak += 1
                    if streak == 4:
                        return True
                else:
                    streak = 0
                pos = pos + direction
        return False


    # Verificar si el juego ya ha acabado
    def is_over(self):
        return (self.board.min() > 0) or self.loss_condition()


    # Computar el score
    def scoring(self):
        return -100 if self.loss_condition() else 0


if __name__ == '__main__':
    # Definir los algoritmos que se estaran utilizando
    algo_neg = Negamax(5)
    algo_sss = SSS(5)

    # Empezar el juego
    game = GameController([AI_Player(algo_neg), AI_Player(algo_sss)])
    game.play()

    # Mostrar en pantalla el resultado
    if game.loss_condition():
        print('\nPlayer', game.nopponent, 'wins.')
    else:
        print("\nIt's a draw.")
```

```
0 1 2 3 4 5 6
-------------
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .

Move #1: player 1 plays 0 :

0 1 2 3 4 5 6
-------------
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
O . . . . . .

Move #2: player 2 plays 0 :

0 1 2 3 4 5 6
-------------
. . . . . . .
. . . . . . .
. . . . . . .
. . . . . . .
X . . . . . .
O . . . . . .

Move #3: player 1 plays 0 :

0 1 2 3 4 5 6
-------------
. . . . . . .
. . . . . . .
. . . . . . .
O . . . . . .
X . . . . . .
O . . . . . .

Move #4: player 2 plays 0 :

0 1 2 3 4 5 6
-------------
. . . . . . .
. . . . . . .
X . . . . . .
O . . . . . .
X . . . . . .
O . . . . . .

Move #5: player 1 plays 0 :

0 1 2 3 4 5 6
-------------
. . . . . . .
O . . . . . .
X . . . . . .
O . . . . . .
X . . . . . .
O . . . . . .

Move #6: player 2 plays 0 :

0 1 2 3 4 5 6
-------------
X . . . . . .
O . . . . . .
X . . . . . .
```

```
O . . . . . .
X . . . . . .
O . . . . . .

Move #7: player 1 plays 1 :

0 1 2 3 4 5 6
-------------
X . . . . . .
O . . . . . .
X . . . . . .
O . . . . . .
X . . . . . .
O O . . . . .

Move #8: player 2 plays 1 :

0 1 2 3 4 5 6
-------------
X . . . . . .
O . . . . . .
X . . . . . .
O . . . . . .
X X . . . . .
O O . . . . .

Move #9: player 1 plays 1 :

0 1 2 3 4 5 6
-------------
X . . . . . .
O . . . . . .
X . . . . . .
O O . . . . .
X X . . . . .
O O . . . . .

Move #10: player 2 plays 1 :

0 1 2 3 4 5 6
-------------
X . . . . . .
O . . . . . .
X X . . . . .
O O . . . . .
X X . . . . .
O O . . . . .

Move #11: player 1 plays 1 :

0 1 2 3 4 5 6
-------------
X . . . . . .
O O . . . . .
X X . . . . .
O O . . . . .
X X . . . . .
O O . . . . .

Move #12: player 2 plays 1 :

0 1 2 3 4 5 6
-------------
X X . . . . .
O O . . . . .
X X . . . . .
O O . . . . .
X X . . . . .
O O . . . . .

Move #13: player 1 plays 2 :
```

```
0 1 2 3 4 5 6
-------------
X X . . . . .
O O . . . . .
X X . . . . .
O O . . . . .
X X . . . . .
O O O . . . .

Move #14: player 2 plays 3 :

0 1 2 3 4 5 6
-------------
X X . . . . .
O O . . . . .
X X . . . . .
O O . . . . .
X X . . . . .
O O O X . . .

Move #15: player 1 plays 2 :

0 1 2 3 4 5 6
-------------
X X . . . . .
O O . . . . .
X X . . . . .
O O . . . . .
X X O . . . .
O O O X . . .

Move #16: player 2 plays 2 :

0 1 2 3 4 5 6
-------------
X X . . . . .
O O . . . . .
X X . . . . .
O O X . . . .
X X O . . . .
O O O X . . .

Move #17: player 1 plays 2 :

0 1 2 3 4 5 6
-------------
X X . . . . .
O O . . . . .
X X O . . . .
O O X . . . .
X X O . . . .
O O O X . . .

Move #18: player 2 plays 2 :

0 1 2 3 4 5 6
-------------
X X . . . . .
O O X . . . .
X X O . . . .
O O X . . . .
X X O . . . .
O O O X . . .

Move #19: player 1 plays 2 :

0 1 2 3 4 5 6
-------------
X X O . . . .
O O X . . . .
X X O . . . .
O O X . . . .
```

```
X X O . . . .
O O O X . . .

Move #20: player 2 plays 3 :

0 1 2 3 4 5 6
-------------
X X O . . . .
O O X . . . .
X X O . . . .
O O X . . . .
X X O X . . .
O O O X . . .

Move #21: player 1 plays 4 :

0 1 2 3 4 5 6
-------------
X X O . . . .
O O X . . . .
X X O . . . .
O O X . . . .
X X O X . . .
O O O X O . .

Move #22: player 2 plays 3 :

0 1 2 3 4 5 6
-------------
X X O . . . .
O O X . . . .
X X O . . . .
O O X X . . .
X X O X . . .
O O O X O . .

Move #23: player 1 plays 3 :

0 1 2 3 4 5 6
-------------
X X O . . . .
O O X . . . .
X X O O . . .
O O X X . . .
X X O X . . .
O O O X O . .

Move #24: player 2 plays 3 :

0 1 2 3 4 5 6
-------------
X X O . . . .
O O X X . . .
X X O O . . .
O O X X . . .
X X O X . . .
O O O X O . .

Move #25: player 1 plays 3 :

0 1 2 3 4 5 6
-------------
X X O O . . .
O O X X . . .
X X O O . . .
O O X X . . .
X X O X . . .
O O O X O . .

Move #26: player 2 plays 4 :

0 1 2 3 4 5 6
```

```
------------
X X O O . . .
O O X X . . .
X X O O . . .
O O X X . . .
X X O X X . .
O O O X O . .

Move #27: player 1 plays 4 :

0 1 2 3 4 5 6
------------
X X O O . . .
O O X X . . .
X X O O . . .
O O X X O . .
X X O X X . .
O O O X O . .

Move #28: player 2 plays 4 :

0 1 2 3 4 5 6
------------
X X O O . . .
O O X X . . .
X X O O X . .
O O X X O . .
X X O X X . .
O O O X O . .

Move #29: player 1 plays 4 :

0 1 2 3 4 5 6
------------
X X O O . . .
O O X X O . .
X X O O X . .
O O X X O . .
X X O X X . .
O O O X O . .

Move #30: player 2 plays 4 :

0 1 2 3 4 5 6
------------
X X O O X . .
O O X X O . .
X X O O X . .
O O X X O . .
X X O X X . .
O O O X O . .

Move #31: player 1 plays 5 :

0 1 2 3 4 5 6
------------
X X O O X . .
O O X X O . .
X X O O X . .
O O X X O . .
X X O X X . .
O O O X O O .

Move #32: player 2 plays 5 :

0 1 2 3 4 5 6
------------
X X O O X . .
O O X X O . .
X X O O X . .
O O X X O . .
X X O X X X .
```

```
O O O X O O .

Move #33: player 1 plays 5 :

0 1 2 3 4 5 6
-------------
X X O O X . .
O O X X O . .
X X O O X . .
O O X X O O .
X X O X X X .
O O O X O O .

Move #34: player 2 plays 5 :

0 1 2 3 4 5 6
-------------
X X O O X . .
O O X X O . .
X X O O X X .
O O X X O O .
X X O X X X .
O O O X O O .

Move #35: player 1 plays 6 :

0 1 2 3 4 5 6
-------------
X X O O X . .
O O X X O . .
X X O O X X .
O O X X O O .
X X O X X X .
O O O X O O O

Move #36: player 2 plays 6 :

0 1 2 3 4 5 6
-------------
X X O O X . .
O O X X O . .
X X O O X X .
O O X X O O .
X X O X X X X
O O O X O O O

Player 2 wins.
```

Artificial Intelligence by Example

Ejercicio 1: Optimizing Blockchains with Naive Bayes

In [ ]:

```
# Leyendo los datos
df = pd.read_csv('https://raw.githubusercontent.com/PacktPublishing/Artificial-Intelligence-By-Example-Second-Edition/master/CH07/data_BC.csv')
df.head()
```

Out[ ]:

	DAY	STOCK	BLOCKS	DEMAND
0	10	1455	78	1
1	11	1666	67	1
2	12	1254	57	1
3	14	1563	45	1

4	DAY	STORE	BLOCKS	DEMAND

In [ ]:

```
# Preparando el set de entrenamiento
X = df.loc[:,'DAY':'BLOCKS']
Y = df.loc[:,'DEMAND']
```

In [ ]:

```
# Eligiendo la clase
clasificador_gaussiano = GaussianNB()
```

In [ ]:

```
# Entrenando el modelo
clasificador_gaussiano.fit(X, Y)
```

Out[ ]:

```
GaussianNB(priors=None, var_smoothing=1e-09)
```

In [ ]:

```
# Predecir con el modelo
print('Bloques para la predicción de la cadena de bloques A-F')

bloques = [[14,1345,12], [29,2034,50], [30,7789,4], [31,6789,4]]
print(bloques)

prediccion = clasificador_gaussiano.predict(bloques)

for i in range(4):
    print(f'Bloque #{i + 1} \nPrediccion Gauss Naive Bayes: {prediccion[i]}')
```

```
Bloques para la predicción de la cadena de bloques A-F
[[14, 1345, 12], [29, 2034, 50], [30, 7789, 4], [31, 6789, 4]]
Bloque #1
Prediccion Gauss Naive Bayes: 1
Bloque #2
Prediccion Gauss Naive Bayes: 2
Bloque #3
Prediccion Gauss Naive Bayes: 2
Bloque #4
Prediccion Gauss Naive Bayes: 2
```

Ejercicio 2: Abstract Image Classification with Convolutional Neural Networks (CNNs)

In [ ]:

```
# rutas de datasets
path_pricipal = '/content/drive/MyDrive/Colab Notebooks/CIC/1 semestre/Matemáticas para l
as ciencias de la computación/datasets/'
A=['dataset_traffic/', 'dataset/']

# referencia a A
escenario = 1

# eleccion de imagenes
directory = path_pricipal + A[escenario]

print("Directorio:", directory)
```

```
Directorio: /content/drive/MyDrive/Colab Notebooks/CIC/1 semestre/Matemáticas para las ci
encias de la computación/datasets/dataset/
```

Parte 1: Creando la CNN

In [ ]:

```
# Entrenando escenarios
# 8000->100
estep = 1000

# 32->10
batchs = 10

# 2000->100
vs = 100

# 25->2
ep = 3
```

In [ ]:

```
# Paso 0: Inicializando la CNN
clasificador =  models.Sequential()

# Paso 1: Convolucion
clasificador.add(layers.Conv2D(32, (3, 3), input_shape = (64, 64, 3),
                               activation = 'relu'))

# Paso 2: Pooling(Puesta en comun)
clasificador.add(layers.MaxPooling2D(pool_size = (2, 2)))

# Paso 3: Agregando una segunda capa convulucional y de pooling
clasificador.add(layers.Conv2D(32, (3, 3), activation = 'relu'))
clasificador.add(layers.MaxPooling2D(pool_size = (2, 2)))

# Paso 4: Flattening (Aplanado)
clasificador.add(layers.Flatten())

# Paso 5: Toda la coneccion (Dense)
clasificador.add(layers.Dense(units = 128, activation = 'relu'))
clasificador.add(layers.Dense(units = 1, activation = 'sigmoid'))

# Paso 6: Optimizador
clasificador.compile(optimizer = 'adam', loss = 'binary_crossentropy',
                     metrics = ['accuracy'])
```

In [ ]:

```
# Resumen de la estructura de la CNN
clasificador.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 62, 62, 32)        896
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 31, 31, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 29, 29, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 128)               802944
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 129
=================================================================
Total params: 813,217
Trainable params: 813,217
Non-trainable params: 0
_________________________________________________________________
```

Parte 2: Ajustando la CNN a las imagenes

In [ ]:

```
# Paso 7: Entrenamiento
entrenar_datagen = tf.compat.v2.keras.preprocessing.image.ImageDataGenerator(
                rescale = 1./255, shear_range = 0.2, zoom_range = 0.2,
                horizontal_flip = True)

# Paso 8: Entrenamiento del set
entrenamiento_set = entrenar_datagen.flow_from_directory(directory + 'training_set',
                                                 target_size = (64, 64),
                                                 batch_size = batchs,
                                                 class_mode = 'binary')

# Paso 9: Puebas
pruebas_datagen = tf.compat.v2.keras.preprocessing.image.ImageDataGenerator(rescale = 1.
/255)

# Paso 10: Pruebas en datos de pruebas
pruebas_set = pruebas_datagen.flow_from_directory(directory + 'test_set',
                                            target_size = (64, 64),
                                            batch_size = batchs,
                                            class_mode = 'binary')

# Paso 11: Entrenamiento
clasificador_final = clasificador.fit_generator(entrenamiento_set,
                         steps_per_epoch = estep , epochs = ep,
                         validation_data = pruebas_set,
                         validation_steps = vs, verbose=2)

print(f'Clasificador: {clasificador_final}')
```

```
Found 20 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
Epoch 1/3
1000/1000 - 112s - loss: 0.0086 - accuracy: 0.9975 - val_loss: 0.0019 - val_accuracy: 1.0
000
Epoch 2/3
1000/1000 - 112s - loss: 4.3511e-09 - accuracy: 1.0000 - val_loss: 0.0015 - val_accuracy:
1.0000
Epoch 3/3
1000/1000 - 113s - loss: 6.9737e-10 - accuracy: 1.0000 - val_loss: 0.0012 - val_accuracy:
1.0000
Clasificador: <tensorflow.python.keras.callbacks.History object at 0x7f6edd3c3d90>
```

In [ ]:

```
# Paso 12: Guardar el modelo
clasificador.save(directory + "model/model3.h5")
print('Entrenamiento terminado, modelo guardado')
```

```
Entrenamiento terminado, modelo guardado
```

Deep Learning Illustrated

Ejercicio 1: Improving Deep Networks - Weight Initialization

Simulamos 784 valores de píxeles como entradas a una única capa densa de neuronas artificiales. La inspiración de nuestra simulación de estas 784 entradas proviene, por supuesto, de nuestros queridos dígitos MNIST. Para el número de neuronas en la capa densa (256), elegimos un número lo suficientemente grande como para que, cuando hagamos algunos gráficos más adelante, tengan datos de sobra

In [3]:

```
n_input = 784
n_dense = 256
```

Inicialización de los parámetros de la red w y b

Antes de empezar a pasar datos de entrenamiento a nuestra red, nos gustaría empezar con parámetros razonablemente escalados. Esto es por dos razones.

1. **Los valores grandes de w y b tienden a corresponder a valores mayores de z y, por tanto, a neuronas saturadas.**
2. **Los valores grandes de los parámetros implicarían que la red tiene una fuerte opinión sobre cómo x está relacionada con y, pero antes de que se produzca cualquier entrenamiento con datos, cualquier opinión fuerte es totalmente inmerecida.**

Por otro lado, los valores de los parámetros de cero implican la opinión más débil sobre la relación entre x e y. Para volver al cuento de hadas, queremos un enfoque intermedio al estilo de Ricitos de Oro, que empiece a entrenar desde un principio equilibrado y aprendible. Con esto en mente, cuando diseñamos la arquitectura de nuestra red neuronal, seleccionamos el método `Zeros()` para inicializar las neuronas de nuestra capa densa con b = 0:

In [4]:

```
b_init = Zeros()
```

Siguiendo la línea de pensamiento del párrafo anterior hasta su conclusión natural, podríamos estar tentados de pensar que también deberíamos inicializar los pesos de nuestra red w con ceros. De hecho, esto sería un desastre de entrenamiento: Si todos los pesos y sesgos fueran idénticos, muchas neuronas de la red tratarían una entrada dada x de forma idéntica, dando al descenso por gradiente estocástico un mínimo de heterogeneidad para identificar los ajustes individuales de los parámetros que podrían reducir el coste C. Sería más productivo inicializar los pesos con un rango de valores diferentes para que cada neurona trate una x dada de forma única, proporcionando así al SGD una amplia variedad de puntos de partida para aproximar y. Por casualidad, algunas de las salidas de las neuronas iniciales pueden contribuir en parte a un mapeo sensato de x a y. Aunque esta contribución será débil al principio, SGD puede experimentar con ella para determinar si podría contribuir a una reducción del coste C entre la predicción y' y el objetivo y.

Como se ha explicado anteriormente, la gran mayoría de los parámetros de una red típica de los parámetros de una red típica son pesos; relativamente pocos son sesgos. Por lo tanto, es aceptable (de hecho, es la práctica más común) inicializar los sesgos con ceros, y los pesos con un rango de valores cercano a cero. Una forma directa de generar valores aleatorios valores cercanos a cero es tomar una muestra de la distribución normal estándar

In [5]:

```
w_init = RandomNormal(stddev= 1.0)
```

In [6]:

```
'''
Para simplificar la actualización más adelante en esta sección, añadimos la
función de activación sigmoide a la capa por separado utilizando
Activation('sigmoid')
'''
model = Sequential()
model.add(Dense(n_dense, input_dim=n_input, kernel_initializer=w_init,
                bias_initializer=b_init))
model.add(Activation('sigmoid'))
```

In [7]:

```
x = np.random.random(( 1,n_input))
```

In [8]:

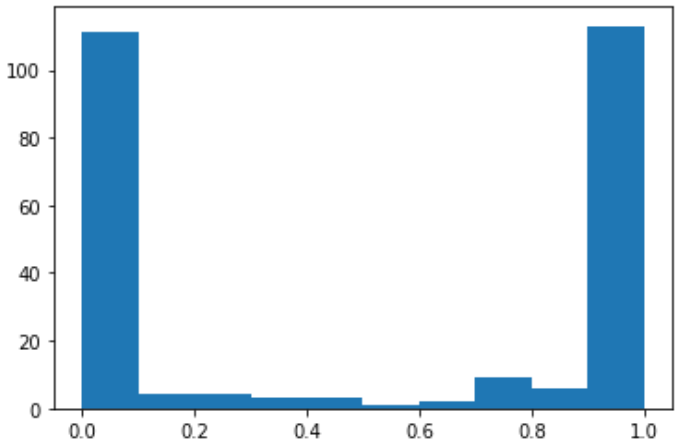
```
'''
Posteriormente, utilizamos el método predict() para propagar x a través de la
capa única y dar salida a las activaciones a:
'''
a = model.predict(x)
```

La visualización muestra que:

1. **Significa que la gran mayoría de las neuronas de la capa están saturadas.**
2. **Implica que las neuronas tienen opiniones firmes sobre cómo influiría x en y antes de cualquier entrenamiento con datos.**

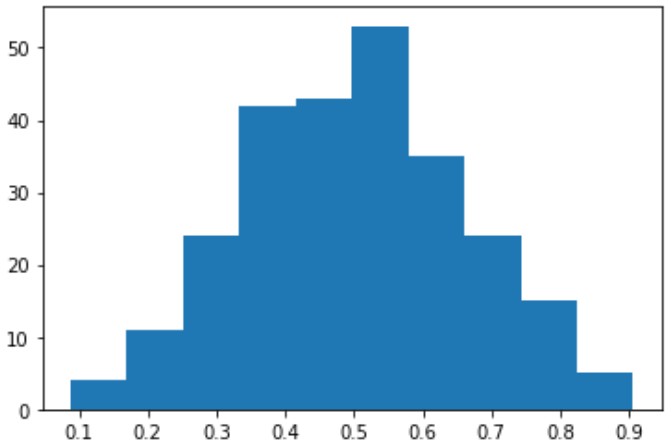
In [9]:

```
# Visualizamos las activaciones
_ = plt.hist(np.transpose(a))
```



In [10]:

```
'''
Repetimos el proceso, cambiando el valor inicial de w a una distribucion normal
de Gorot, y vemos que cambia el grafico
'''

w_init = glorot_normal()

model = Sequential()
model.add(Dense(n_dense, input_dim=n_input, kernel_initializer=w_init,
                bias_initializer=b_init))
model.add(Activation('sigmoid'))

x = np.random.random(( 1,n_input))

a = model.predict(x)

_ = plt.hist(np.transpose(a))
```



Ejercicio 2: Improving Deep Networks - Regression

Llamando al parámetro de forma de `X_train` y `X_valid`, encontramos que hay 404 casos de entrenamiento y 102 casos de validación. Para cada caso -una zona distinta de los suburbios de Boston- tenemos 13 variables predictoras relacionadas con la antigüedad de los edificios, el número medio de habitaciones, el índice de

criminalidad, la proporción local de alumnos por profesor, etc. El precio medio de la vivienda (en miles de dólares) de cada zona se proporciona en las variables y.

Como ejemplo, el primer caso del conjunto de entrenamiento tiene un precio medio de la vivienda de 15.200.32 dólares

In [21]:

```
(X_train, y_train), (X_valid, y_valid) = boston_housing.load_data()
```

Razonando que con sólo 13 valores de entrada y unos pocos cientos de casos de entrenamiento ganaríamos poco con una red neuronal profunda con montones de neuronas en cada capa, optamos por una arquitectura de dos capas ocultas que consta de sólo 32 y 16 neuronas por capa. Aplicamos la normalización por lotes y un toque de abandono para evitar el sobreajuste a los casos particulares del conjunto de datos de entrenamiento. Lo más importante es que en la capa de salida establecemos el argumento de activación como lineal, que es la opción que hay que elegir cuando se quiere predecir una variable continua, como hacemos al realizar una regresión. La función de activación lineal da salida a z directamente, de modo que ^y de la red puede ser cualquier valor numérico en lugar de ser aplastado en una probabilidad entre 0 y 1

In [22]:

```
model = Sequential()
model.add(Dense( 32, input_dim= 13, activation= 'relu' ))
model.add(BatchNormalization())
model.add(Dense( 16, activation= 'relu'))
model.add(BatchNormalization())
model.add(Dropout( 0.2 ))
model.add(Dense( 1 , activation= 'linear' ))
```

Al compilar el modelo, otro ajuste específico de la regresión que realizamos es el uso del error cuadrático medio (MSE) en lugar de la entropía cruzada (loss='mean_squared_error'). Aunque hasta ahora hemos utilizado exclusivamente el coste de entropía cruzada en este libro, esa función de coste está diseñada específicamente para problemas de clasificación, en los que ^y es una probabilidad. Para los problemas de regresión, en los que la salida no es inherentemente una probabilidad, utilizamos el MSE en su lugar.

In [23]:

```
model.compile(loss= 'mean_squared_error' , optimizer= 'adam')
```

Entrenamos durante 32 épocas porque, en nuestra experiencia con este modelo en particular, entrenar durante más tiempo no produjo menores pérdidas de validación. No dedicamos ningún tiempo a optimizar el hiperparámetro del tamaño del lote, por lo que podría haber pequeñas ganancias de precisión al variarlo.

In [24]:

```
model.fit(X_train, y_train, batch_size= 8 , epochs=32 , verbose= 1 ,
          validation_data=(X_valid, y_valid))
```

```
Epoch 1/32
51/51 [==============================] - 1s 6ms/step - loss: 589.6874 - val_loss: 719.201
2
Epoch 2/32
51/51 [==============================] - 0s 3ms/step - loss: 533.8757 - val_loss: 647.380
7
Epoch 3/32
51/51 [==============================] - 0s 3ms/step - loss: 523.1204 - val_loss: 521.472
9
Epoch 4/32
51/51 [==============================] - 0s 4ms/step - loss: 523.2732 - val_loss: 473.287
7
Epoch 5/32
51/51 [==============================] - 0s 4ms/step - loss: 474.5257 - val_loss: 385.770
0
Epoch 6/32
51/51 [==============================] - 0s 3ms/step - loss: 450.1852 - val_loss: 363.133
1
Epoch 7/32
```

```
Epoch 7/32
51/51 [==============================] - 0s 3ms/step - loss: 375.8132 - val_loss: 314.888
0
Epoch 8/32
51/51 [==============================] - 0s 4ms/step - loss: 401.6508 - val_loss: 359.748
8
Epoch 9/32
51/51 [==============================] - 0s 3ms/step - loss: 348.8830 - val_loss: 347.194
2
Epoch 10/32
51/51 [==============================] - 0s 3ms/step - loss: 308.5303 - val_loss: 329.085
6
Epoch 11/32
51/51 [==============================] - 0s 3ms/step - loss: 276.5966 - val_loss: 260.679
1
Epoch 12/32
51/51 [==============================] - 0s 4ms/step - loss: 212.9193 - val_loss: 149.906
4
Epoch 13/32
51/51 [==============================] - 0s 3ms/step - loss: 198.8243 - val_loss: 146.350
6
Epoch 14/32
51/51 [==============================] - 0s 3ms/step - loss: 169.1615 - val_loss: 173.018
4
Epoch 15/32
51/51 [==============================] - 0s 4ms/step - loss: 140.1814 - val_loss: 101.022
2
Epoch 16/32
51/51 [==============================] - 0s 3ms/step - loss: 115.8618 - val_loss: 112.674
5
Epoch 17/32
51/51 [==============================] - 0s 4ms/step - loss: 101.3823 - val_loss: 131.940
4
Epoch 18/32
51/51 [==============================] - 0s 3ms/step - loss: 108.5526 - val_loss: 132.229
9
Epoch 19/32
51/51 [==============================] - 0s 3ms/step - loss: 65.0686 - val_loss: 111.6780
Epoch 20/32
51/51 [==============================] - 0s 4ms/step - loss: 46.5208 - val_loss: 68.7498
Epoch 21/32
51/51 [==============================] - 0s 3ms/step - loss: 57.2931 - val_loss: 31.1208
Epoch 22/32
51/51 [==============================] - 0s 3ms/step - loss: 52.4971 - val_loss: 77.3134
Epoch 23/32
51/51 [==============================] - 0s 4ms/step - loss: 45.2600 - val_loss: 55.0409
Epoch 24/32
51/51 [==============================] - 0s 3ms/step - loss: 39.9543 - val_loss: 31.8873
Epoch 25/32
51/51 [==============================] - 0s 3ms/step - loss: 50.2973 - val_loss: 33.1460
Epoch 26/32
51/51 [==============================] - 0s 3ms/step - loss: 39.2941 - val_loss: 27.9414
Epoch 27/32
51/51 [==============================] - 0s 4ms/step - loss: 40.4787 - val_loss: 32.6546
Epoch 28/32
51/51 [==============================] - 0s 4ms/step - loss: 35.2051 - val_loss: 29.9324
Epoch 29/32
51/51 [==============================] - 0s 3ms/step - loss: 38.4100 - val_loss: 44.4306
Epoch 30/32
51/51 [==============================] - 0s 3ms/step - loss: 44.0419 - val_loss: 50.3255
Epoch 31/32
51/51 [==============================] - 0s 3ms/step - loss: 37.3613 - val_loss: 32.5703
Epoch 32/32
51/51 [==============================] - 0s 3ms/step - loss: 48.3042 - val_loss: 65.7735
```

Out[24]:

```
<tensorflow.python.keras.callbacks.History at 0x7f5062578190>
```

Esto nos devolvió una predicción del precio medio de la vivienda (^y) de 22,082 dólares para el suburbio 43 de Boston en el conjunto de datos de validación. El precio medio real (y; que puede obtenerse llamando a `y_valid[42]`) es de 14,100 dólares.

In [25]:

```
model.predict(np.reshape(X_valid[42], [1 , 13]))
```

Out[25]:

```
array([[22.082865]], dtype=float32)
```

In [26]:

```
y_valid[42]
```

Out[26]:

```
14.1
```