Departamento de Ciencia de la Computación e Inteligencia Artificial
Universidad de Alicante v2

Práctica 2. Visión Artificial y Aprendizaje

Convocatoria C4

Objetivos:

- Comprender el funcionamiento del aprendizaje automático supervisado y en concreto las redes neuronales multicapa (MLP).
- Entender el funcionamiento de la red y como realiza sus cálculos en detalle
- Implementar la fase forward para una red MLP concreta
- Entender el concepto de imagen y píxel, y cómo podemos aprender a distinguir entre imágenes a partir de la información que aparece en ellas.
- Utilizar una librería de Deep Learning para aprender a clasificar dígitos manuscritos.
- Realizar un análisis cuantitativo de la tasa de aciertos obtenida mediante este método de aprendizaje

En esta práctica trabajaremos en dos partes. La primera versará sobre el funcionamiento a bajo nivel de una red neuronal, de terminando detalladamente el funcionamiento de una red multicapa para aprender una función booleana. A continuación, utilizaremos la librería de Deep Learning (*Keras)* para aprender un problema más completo: aprender a clasificar 10 dígitos manuscritos.

NOTA que: en esta convocatoria los apartados I4 e II4 pasan a ser obligatorios contando 5 puntos sobre el total de la práctica.

Parte I. Aprende las bases de un MLP

[2,5pts sin contar el apartado 4]

Todo el código que desarrolles para el cálculo de cualquier apartado de esta parte de la práctica debe encontrarse en el fichero **`p2base.py`** (todo en minúsculas).

Ten en cuenta que el código incluido debe de ser ejecutable con resultados reproducibles sin necesidad de comentar o descomentar grandes extensiones de código (organízalo en funciones y usa `if __name__ == "__main__":` para posibilitar elegir que código lanzar).

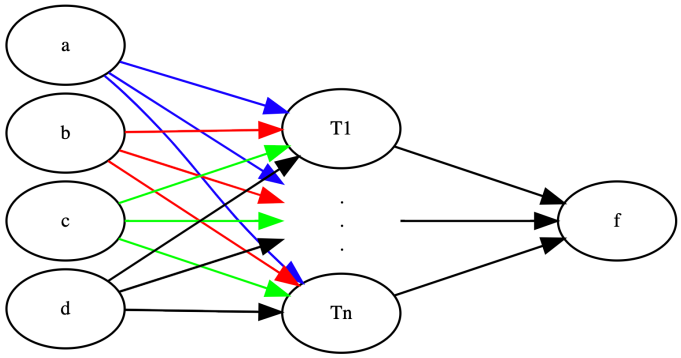
El fichero debe estar organizado en el mismo orden que este enunciado. Se valorará la organización y documentación del código generado. **En la documentación referencia que hay que hacer para repetir los cálculos que has realizado para cada pregunta.**

I1) Resolviendo una función booleana mediante un "Multi Layer Perceptron" (MLP)

Notar que:

Cada estudiante tendrá asignada una función booleana distinta que debe conocer antes de realizar la práctica.

Se dispone de una función booleana de la forma $f = T_1 \vee T_2 \vee \ldots \vee T_n$. Las entradas de la red serán las variables a, b, c, d y su salida un valor booleano que debe computar la función asignada. Puede ser razonable que cada término T_n corresponda con una neurona de la primera capa oculta, tal que:



- A) Diseña una red MLP con funciones de activación sigmoidea que resuelva tu función booleana. **Importante: no puedes simplificar la función, debes computarla con todos los términos T_n que contenga (aunque por ejemplo tengas alguno duplicado)**
  - Debes calcular los pesos y umbrales de activación que debe tener la red. Puedes utilizar una calculadora (o Python, que es lo que recomendamos) pero no una librería neuronal para su cálculo.
  - De hecho**, te pedimos que expliques con detalle que proceso sigues para conseguir ajustar manualmente los pesos**.
- B) Implementa en python una función `y = forward((a,b,c,d))` que reciba como parámetro **una tupla** de componentes (a, b, c, d) y calcule la fase *forward* de tu red. **Solo se pueden utilizar las librerías** `math` y `numpy` de python para este apartado.
  - Esta función debe llamarse `forward` y debe recibir y retornar los parámetros indicados. No debe imprimir nada por la salida estándar
  - Si incluimos tu fichero de esta parte `p2base.py` debemos de poder gastar tu función. Verifícalo:

```
from p2base import forward
i = (0,0,0,1)
print(forward(i)) #Debe emitir un número real como salida exclusivamente
```

I2) Modelar, entrenar y probar la red en Keras

Ahora vamos a utilizar Keras para el proceso de entrenamiento. Para ello seguiremos los siguientes pasos.

1. Crea una rutina que evalúe tu función booleana para todo el dominio (todas las combinaciones de a, b, c, d). Guarda los resultados en dos vectores X (entrada) e Y (salida booleana de la función).
2. Diseña una red idéntica al MLP anterior en keras. Utiliza como función de coste/pérdida/error MSE y `Adam` como algoritmo de optimización. Tu código será parecido a este, ajustando el número de capas y las neuronas por capa (debes sustituir el '`?`'):

```
from tensorflow import keras
from tensorflow.keras import layers

model = keras.Sequential(
    [
        keras.Input(shape=(?)),
        layers.Dense(?, activation="sigmoid"),
        ...
        layers.Dense(?, activation="sigmoid"),
    ]
)
model.compile(loss="mean_squared_error", optimizer="adam")
```

3. Entrena la red. Eso lo debes hacer mediante la función `model.fit` de keras. Determina que `batch_size` es conveniente utilizar y el número de épocas (`epochs`) para que el entrenamiento sea fructífero.
4. Comprueba cómo funciona la red con tus conjuntos X, Y. Debes utilizar la función de keras `model.predict` y determinar cuál es la tasa de acierto de tu red para tu función booleana.

I3) Analizar el entrenamiento y comparar con la red ajustada a mano

Contesta de manera razonada a las siguientes preguntas:

- ¿Cuántos pasos de entrenamiento necesitas para que aprenda correctamente la función? ¿Te parecen muchos?
- ¿Qué es el parámetro *loss* y *optimizer* de la red? ¿Crees que afectan al entrenamiento? ¿Qué es el *learning rate* (LR) en una red neuronal? ¿Se puede ajustar el LR en Keras? Haz pruebas para intentar minimizar el número de pasos de entrenamiento requeridos. Comenta los resultados obtenidos
- Compara los pesos y bias aprendidos con los que pusiste a mano ¿Se parecen? ¿Ha aprendido la misma función? Justifica tu respuesta. Ten en cuenta que puedes utilizar el siguiente código para acceder a los pesos del modelo una vez aprendido:

```
first_layer_weights = model.layers[0].get_weights()[0]
first_layer_biases = model.layers[0].get_weights()[1]
second_layer_weights = model.layers[1].get_weights()[0]
second_layer_biases = model.layers[1].get_weights()[1]
...
```

I4) Aplica backpropagation manualmente [obligatorio, 2.5pts]

Aplica backpropagation **manualmente** a tu red para un par de entradas de entrenamiento y explica su funcionamiento de manera clara y detallada. Se debe hacer una traza paso a paso, dado unos pesos determinados, explicando cómo se realizan los cálculos y como se ajustan los pesos. Se valorará la didáctica utilizada para explicar convenientemente el proceso realizado así como la presentación del apartado. Comprueba que el ajuste mejora el desempeño de la red. No es necesario implementar el algoritmo.

Parte II. Entrena un MLP mediante Deep Learning usando Keras [2.5pts sin contar el apartado 4]

En este apartado aprenderemos a desarrollar un clasificado de dígitos manuscritos. Para ello utilizaremos una base de datos de dígitos manuscritos y una librería avanzada de deep learning.

Todo el código que desarrolles para el cálculo de cualquier apartado de esta parte de la práctica debe encontrarse en el fichero **`p2mnist.py`** (todo en minúsculas).

Ten en cuenta que el código incluido debe de ser ejecutable con resultados reproducibles sin necesidad de comentar o descomentar grandes extensiones de código (organizalo en funciones y usa `if __name__ == "__main__":` para posibilitar elegir que código lanzar).

El fichero debe estar organizado en el mismo orden que este enunciado. Se valorará la organización y documentación del código generado. **En la documentación referencia que hay que hacer para repetir los cálculos que has realizado para cada pregunta.**

II1) Procesamiento de los datos

a) Busca información sobre el conjunto/base de datos MNIST (*http://yann.lecun.com/exdb/mnist/*) y explica con detalle para que sirve y su importancia. Una vez que hayas comprendido su estructura cárgalo utilizando la librería para *deep learning* keras (*https://keras.io/*). Esta librería ya incluye una directiva sencilla para cargar el conjunto: `keras.datasets.mnist.load_data()` (puedes utilizarla).

b) Dado el formato de la base de datos MNIST para clasificar dígitos manuscritos contesta las siguientes preguntas de **manera detallada y razonada**:

- ¿Cuántas neuronas necesitamos en la capa de entrada? ¿Y en la capa de salida?
- ¿Cómo dividirás el conjunto de entrenamiento/test/validación?
- ¿Cómo preprocesaras los datos de entrada?
- ¿Cómo transformarás la imagen para poder entrenarla con una red MLP?

II2) Implementa la red en keras

Implementa la red en keras. Para ello debes utilizar exclusivamente capas de tipo `Dense`, ya que deseamos desarrollar la red como un *vanilla MLP*. Utilizaremos funciones de activación `relu` para las capas ocultas y `softmax` para la capa de salida. Gastaremos dos capas ocultas con `128` neuronas. Por tanto, el diseño de la red será parecido a:

```
model = keras.Sequential(
    [
        keras.Input(shape=______),
        layers.Dense(128, activation="relu"),
        layers.Dense(128, activation="relu"),
        layers.Dense(______, activation="softmax"),
    ]
)
```

c) Contesta detalladamente a las siguientes preguntas:

- ¿Qué es la función de activación `relu`? ¿Cómo varia de la sigmoidea vista en teoría?

- ¿Qué consigue la función de activación `softmax` de la última capa? ¿Se podría usar una función de activación `relu` en la última capa?
- ¿Qué función de activación crees que es mejor? ¿Por qué crees que se suelen utilizar más `relu` que su alternativa `sigmoidea`?

Ahora compila el modelo y entrénalo. Utilizaremos como función de error la entropía cruzada categórica y como algoritmo de optimización `adam`. Puedes usar este código como referencia:

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

model.fit(x_train, y_train, batch_size=128,
          epochs=15, validation_split=0.1)
```

d) Contesta razonadamente a las siguientes preguntas

- Explica que son y para qué sirven los parámetros `batch_size` y `validation_split`.
- ¿Conoces algún otro algoritmo de optimización distinto `adam`? Comenta brevemente el funcionamiento de otro distinto.

II3) Prueba el modelo

Prueba el modelo. Keras permite utilizar cualquier modelo entrenado de manera sencilla.

e) Una vez entrenado el modelo contesta a las siguientes preguntas:

- ¿Qué error de clasificación obtienes para los datos de test? ¿Y qué valor de perdida de la función de coste? *Para este apartado gasta la función de keras `model.evaluate`*
- Escoge 1000 elementos al azar del conjunto de entrenamiento y comprueba que tasa de error de clasificación obtienes. ¿Coincide con la métrica anterior?
- Escoge 1000 elementos al azar del conjunto de test y comprueba que tasa de error de clasificación obtienes. ¿Coincide con la métrica anterior?

II4) Mejora la red [obligatorio, 2.5pts]

Intentar mejorar el resultado de la clasificación cambiando la red o su entrenamiento. Se permiten utilizar cualquier mejora, pero:

a) Se deben realizar varias pruebas con una red un *vanilla MLP*. Se puede sacar la máxima nota haciendo pruebas con este tipo de red en exclusiva.

b) Se pueden probar además otras arquitecturas más adaptadas al uso de imágenes. En este caso se deben realizar varias pruebas para mejorar lo máximo posible la arquitectura escogida (a parte de las del apartado a).

Debes explicar las técnicas utilizadas, el proceso seguido y la mejora conseguida con respecto al conjunto de test. Se debe informar tanto de las pruebas realizadas que no consigan mejoría como las que sí. Se valorará el número de pruebas realizada, la justificación de porqué se han realizado las pruebas (y que estrategia se ha seguido para diseñarlas) así como las referencias bibliográficas utilizadas.

Cuestiones finales

Documentación y pruebas. Hito final

Para este hito final debemos tener realizadas diferentes pruebas y terminar la documentación que se deberá haber ido elaborando de manera continua durante todo el desarrollo de la práctica. La documentación **es la parte más importante de la práctica (60%)** y en ella se debe realizar un análisis minucioso de la misma reflejando de manera clara y razonada, contestando todas las preguntas anteriores.

Entrega de la práctica

La fecha límite de entrega de la práctica es el **06 de Julio de 2023 a las 23:55h**. La entrega se realizará a través de Moodle.

Formato para la entrega final

La entrega debe consistir en un fichero comprimido zip con:

- Dos ficheros **fuentes** con todo el código necesario para desarrollar las dos partes de la práctica. Todo lo documentado debe ser reproducible ejecutando el código que entreguéis en dos ficheros: **`p2base.py y p2mnist.py`**
- Fichero de **documentación** en formato **pdf (**no se leerá otro formato)**,** que contiene toda la documentación, respuestas a las preguntas planteadas en el enunciado y hace referencia a las pruebas realizadas en los dos ficheros anteriores y cómo reproducirlas.

¡¡¡AVISO IMPORTANTE!!!

No cumplir cualquiera de las normas de formato/entrega puede suponer un suspenso en la práctica.

Recordad que **las prácticas son INDIVIDUALES** y NO se pueden hacer en parejas o grupos.

Cualquier código copiado supondrá un suspenso de la práctica para todas las personas implicadas en la copia y, como indica el Reglamento para la Evaluación de Aprendizajes de la Universidad de Alicante (BOUA 9/12/2015) y el documento de Actuación ante copia en pruebas de evaluación de la EPS, **se informará a la dirección de la Escuela Politécnica Superior para la toma de medidas oportunas**.

La nota de la práctica sufrirá una penalización de dos puntos si no se cumple rigurosamente con los requisitos de la entrega (tanto en la estructura de los ficheros entregados como en la salida que debe generar la práctica).