Universidad Politécnica de Madrid

Herramientas software para la realización de proyectos de AI aplicados a la investigación

Proyecto de IA basado en Pytorch

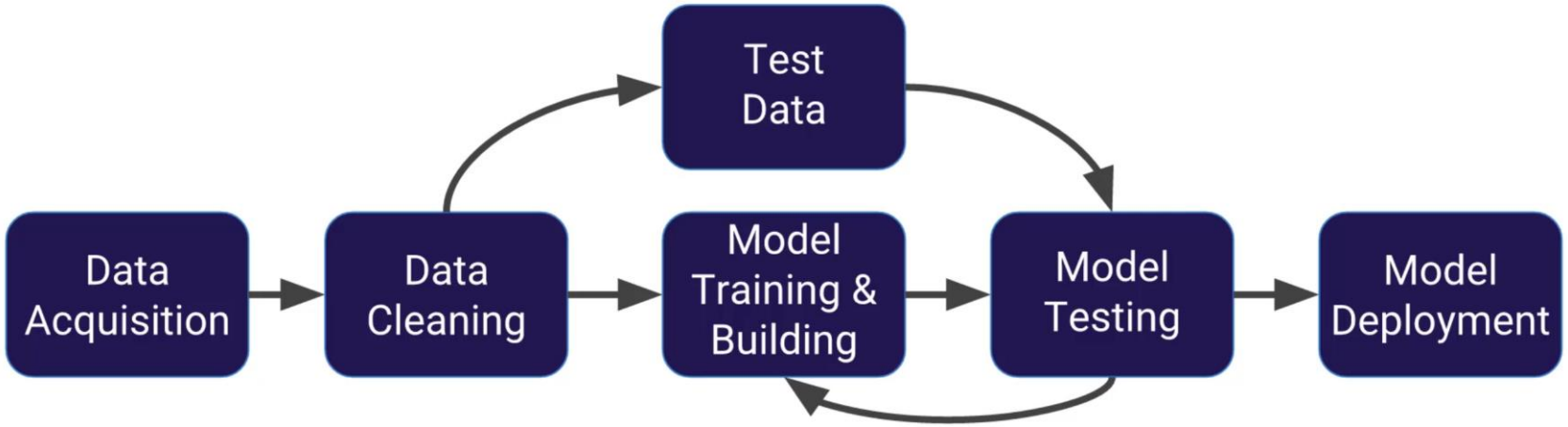
Alberto Belmonte Hernández

Índice

- Fases en un proyecto de IA
- Python y Deep Learning
- Visión general de un proyecto DL con Pytorch
  - Instalación de Pytorch
  - Ciclo de vida del modelo de aprendizaje profundo de PyTorch
  - Cómo desarrollar modelos de aprendizaje profundo en PyTorch
    - Cómo desarrollar un MLP para la clasificación binaria
    - Cómo desarrollar un MLP para la clasificación multiclase
    - Cómo desarrollar un MLP para la regresión
    - Cómo desarrollar una CNN para la clasificación de imágenes

Fases en un proyecto de IA

- En el primer módulo del curso presentamos la figura que vemos en esta diapositiva. En este módulo vamos a realizar todas las partes implicadas ya que son relativas a un proyecto de IA. Crearemos un proyecto en el que partiremos de un dataset, realizaremos la limpieza, crearemos y entrenaremos un modelos de Deep Learning y lo testearemos.



- Para todo el desarrollo vamos a emplear el lenguaje de programación Python y para la parte del modelo de Deep Learning emplearemos Pytorch que es una librería de Python que nos permite realizar tareas de aprendizaje profundo de manera automática.
- En este módulo nos vamos a centrar principalmente en la estructura y partes que debe contener el código y que seguirán los módulos que hemos visto en la figura. No preocuparse si no sois expertos en Python o en Deep Learning pues la idea principal es ver la estructura del código y las tareas a realizar.
- Tras este módulo, la ideas principales expuestas pueden ser usadas para cualquier otro proyecto de IA siguiendo el mismo esquema que vamos a realizar y adaptando cada parte (dataset, limpieza, modelo, parámetros de entrenamiento…) a los datos y requerimientos del proyecto.

Python y Deep Learning

- En la actualidad, Python se ha convertido en el lenguaje más popular entre la comunidad científica y desarrolladora para temas relacionados con la inteligencia artificial. Han surgido numerosas librerías para crear proyectos de este tipo y la facilidad de uso de librerías para el empleo de entrenamientos en GPU han provocado un "boom" en este campo de estudio.
- Esto no quiere decir que no se pueda hacer IA en otros lenguajes, existen gran cantidad de librerías dedicadas para lenguajes como C#, C++, JAVA, Javascript…
- Dentro de Python las librerías fundamentales para IA son Sklearn (ML), Keras, Tensorflow y Pytorch (DL), entre otras, pero menos populares.
- En este módulo nos vamos a centrar en Pytorch, por lo que vamos a comentar muy brevemente las características de este frente a los demás, y algunas diferencias.

	API Level	Arquitectura	Datasets	Debugging	Trained Models	Velocidad	Escrito en
Keras	Alto	Simple, concisa y leíble	Datasets pequeños	Redes sencillas, no necesidad	Si	Lento, baja performance	Python
Pytorch	Bajo	Compleja, menos fácil de leer	Datasets grandes	Buenas capacidades para Debug	Si	Rápido, buena performance	Lua
Tensorflow	Alto y Bajo	No es fácil de usar	Datasets grandes	Dificil realizar Debug	Si	Rápido, buena performance	C++, CUDA, Python

Visión general de un proyecto DL con Pytorch

- El enfoque de este tutorial es el uso de la API de PyTorch para tareas comunes de desarrollo de modelos de aprendizaje profundo. No nos sumergiremos en las matemáticas y la teoría del aprendizaje profundo. La mejor manera de aprender deep learning en python es haciendo. Se ha diseñado cada ejemplo de código para utilizar las mejores prácticas y ser independiente para que pueda copiar y pegar directamente en su proyecto y adaptarlo a sus necesidades específicas. Esto te dará una gran ventaja sobre el intento de entender la API sólo con la documentación oficial.

- El tutorial se centrará en la instalación, ciclo de vida de un modelo en Pytorch y diferentes ejemplos de problemas a resolver con Deep Learning.

- No es necesario que lo entiendas todo (al menos no en este momento). Su objetivo es recorrer el tutorial de principio a fin y obtener un resultado. No es necesario que lo entiendas todo a la primera. Utiliza mucho la documentación de la API para aprender sobre todas las funciones que estás utilizando.

- No es necesario que conozcas las matemáticas a la primera. Las matemáticas son una forma compacta de describir el funcionamiento de los algoritmos, concretamente las herramientas del álgebra lineal, la probabilidad y el cálculo. Estas no son las únicas herramientas que puedes utilizar para aprender cómo funcionan los algoritmos. También puedes usar código y explorar el comportamiento de los algoritmos con diferentes entradas y salidas. Conocer las matemáticas no te dirá qué algoritmo elegir o cómo configurarlo mejor. Eso sólo lo puedes descubrir a través de experimentos cuidadosamente controlados.

- No es necesario saber cómo funcionan los algoritmos. Es importante conocer las limitaciones y la forma de configurar los algoritmos de aprendizaje profundo. Pero el aprendizaje de los algoritmos puede venir después. Necesitas construir este conocimiento de los algoritmos lentamente durante un largo periodo de tiempo. Hoy, empieza por sentirte cómodo con la plataforma.

- No es necesario que seas un programador de Python. La sintaxis del lenguaje Python puede ser intuitiva si eres nuevo en él. Al igual que otros lenguajes, céntrate en las llamadas a funciones (por ejemplo, function()) y en las asignaciones (por ejemplo, a = "b"). Esto te servirá para la mayor parte del camino. Eres un desarrollador, sabes cómo coger los fundamentos de un lenguaje realmente rápido. Sólo tienes que empezar y sumergirte en los detalles más tarde.

Instalación de Pytorch

- Para instalar Pytorch puedes ir a a web propia dónde se encuentran todas las versiones disponibles, tanto para diferentes sistemas operativos, versiones de CUDA o lenguajes de programación:

  https://pytorch.org/get-started/locally/

- Dado que trabajaremos en Python, buscaremos la versión adecuada mediante "pip" o "conda" (el que se tenga disponible) y elegiremos la versión CUDA o CPU dependiendo de si disponemos de tarjeta gráfica y entorno CUDA o queremos ejecutar todo en el procesador del PC. En el entorno seleccionable de la web escogemos lo deseado y nos aparecerá la línea de comando que debemos emplear. En mi caso, dispongo de un PC con una NVIDIA GTX 2070 y entorno CONDA. Quiero trabajar con la versión CUDA 11.3 por tanto el comando final será:

  conda install pytorch torchvision torchaudio cudatoolkit=11.3 -c pytorch

- Esto nos insalará todos los paquetes de pytorch incluido librerías adicionales para procesado de imagen como torchvision y para audio como torchaudio así como lo necesario para usar CUDA y la GPU de mi ordenador.

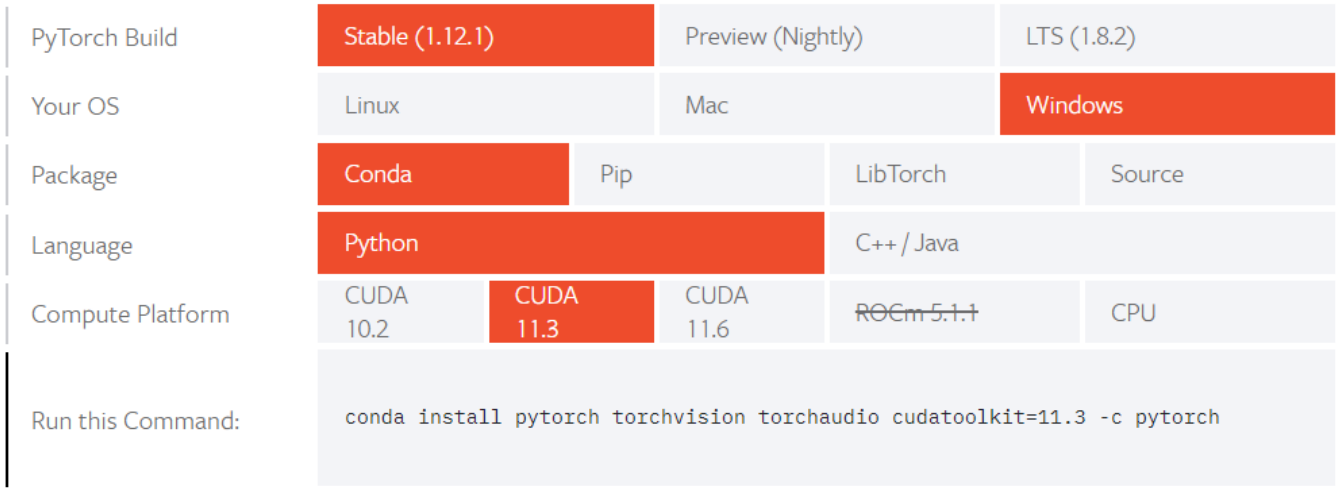
- Para comprobar que lo tenemos instalado podemos entrar a Python desde un terminal e importar el paquete mediante "import torch".
- Podemos además comprobar que Pytorch detecta la GPU con los siguientes comandos:

  torch.cuda.is_available()

  torch.cuda.device_count()

  torch.cuda.current_device()

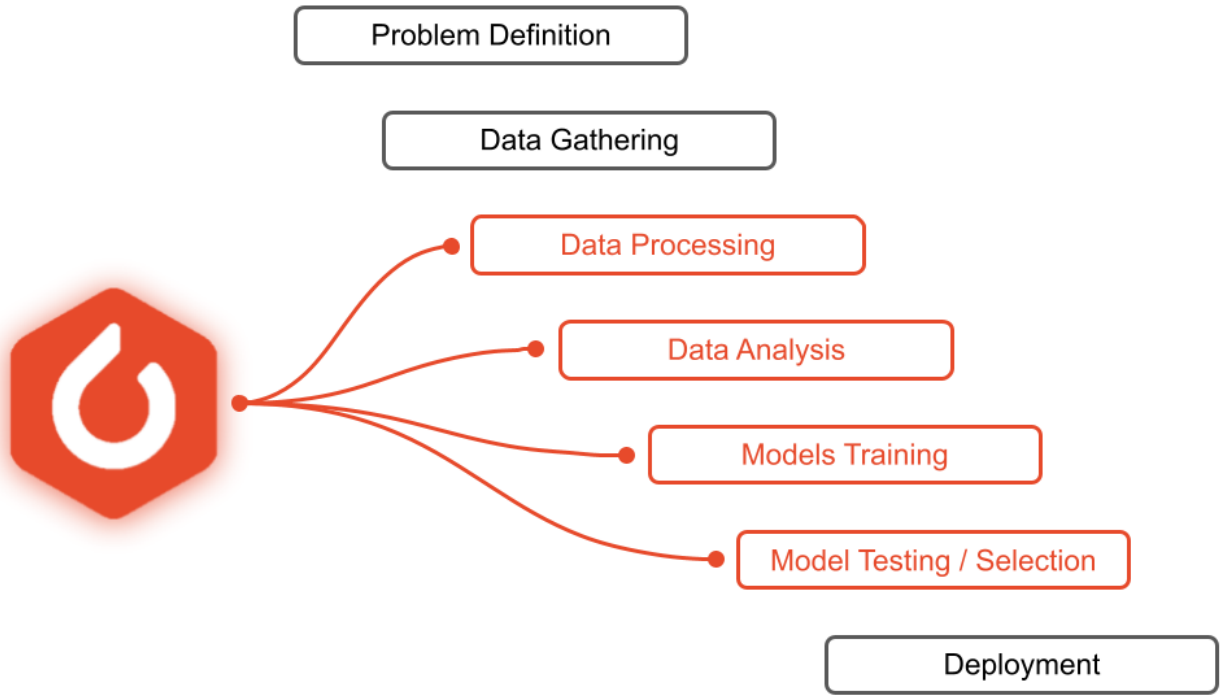
  torch.cuda.get_device_name(0)



Ciclo de vida del modelo de aprendizaje profundo de PyTorch

- En esta parte se presentará el ciclo de vida de un modelo de aprendizaje profundo y la API de PyTorch que puedes utilizar para definir modelos. Un modelo tiene un ciclo de vida, y este conocimiento muy simple proporciona la columna vertebral tanto para modelar un conjunto de datos como para entender la API de PyTorch.
- Los cinco pasos del ciclo de vida son los siguientes:
  - 1. Preparar los datos.
  - 2. Definir el modelo.
  - 3. Entrenar el modelo.
  - 4. Evaluar el modelo.
  - 5. Hacer predicciones.

- Nota: Hay muchas maneras de lograr cada uno de estos pasos utilizando la API de PyTorch, aunque se mostrará lo más simple, o más común, o más idiomático.
- Como se observa el ciclo de vida es similar al de la figura presentada al inicio del módulo incluyendo todas las partes ya comentadas que veremos ahora como realizarlas mediante código.



Paso 1: Preparar los datos

- El primer paso es cargar y preparar los datos. Los modelos de redes neuronales requieren datos numéricos de entrada y datos numéricos de salida.
- Puede utilizar las bibliotecas estándar de Python para cargar y preparar los datos tabulares, como los archivos CSV. Por ejemplo, Pandas se puede utilizar para cargar su archivo CSV, y las herramientas de scikit-learn se pueden utilizar para codificar los datos categóricos, como las etiquetas de clase.
- PyTorch proporciona la clase Dataset que puedes extender y personalizar para cargar tu conjunto de datos.
- Por ejemplo, el constructor de tu objeto dataset puede cargar tu archivo de datos (por ejemplo, un archivo CSV). A continuación, puede anular la función __len__() que puede utilizarse para obtener la longitud del conjunto de datos (número de filas o muestras), y la función __getitem__() que se utiliza para obtener una muestra específica por índice.
- Al cargar el conjunto de datos, también se pueden realizar las transformaciones necesarias, como el escalado o la codificación.
- A continuación se proporciona un esqueleto de una clase Dataset personalizada.

```
# dataset definition
class CSVDataset(Dataset):
    # load the dataset
    def __init__(self, path):
        # store the inputs and outputs
        self.X = ...
        self.y = ...

    # number of rows in the dataset
    def __len__(self):
        return len(self.X)

    # get a row at an index
    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]
```

Paso 1: Preparar los datos

- Una vez cargado, PyTorch proporciona la clase DataLoader para navegar por una instancia de Dataset durante el entrenamiento y la evaluación de su modelo.
- Se puede crear una instancia de DataLoader para el conjunto de datos de entrenamiento, el conjunto de datos de prueba e incluso un conjunto de datos de validación.
- La función random_split() puede utilizarse para dividir un conjunto de datos en conjuntos de entrenamiento y de prueba. Una vez dividido, se puede proporcionar una selección de filas del conjunto de datos a un DataLoader, junto con el tamaño del lote y si los datos deben ser barajados cada época.
- Por ejemplo, podemos definir un DataLoader pasando una muestra seleccionada de filas del conjunto de datos.

```
...
# create the dataset
dataset = CSVDataset(...)
# select rows from the dataset
train, test = random_split(dataset, [[...], [...]])
# create a data loader for train and test sets
train_dl = DataLoader(train, batch_size=32, shuffle=True)
test_dl = DataLoader(test, batch_size=1024, shuffle=False)
```

- Una vez definido, un DataLoader puede ser enumerado, produciendo un lote de muestras en cada iteración.

```
...
# train the model
for i, (inputs, targets) in enumerate(train_dl):
    ...
```

Paso 2: Definir el modelo

- El siguiente paso es definir un modelo. El lenguaje para definir un modelo en PyTorch implica definir una clase que extienda la clase Module.
- El constructor de tu clase define las capas del modelo y la función forward() es el override que define cómo propagar la entrada a través de las capas definidas del modelo.
- Hay muchas capas disponibles, como Linear para las capas totalmente conectadas, Conv2d para las capas convolucionales y MaxPool2d para las capas de agrupación.
- Las funciones de activación también pueden definirse como capas, como ReLU, Softmax y Sigmoid.
- A continuación se muestra un ejemplo de un modelo MLP simple con una capa.

```
# model definition
class MLP(Module):
    # define model elements
    def __init__(self, n_inputs):
        super(MLP, self).__init__()
        self.layer = Linear(n_inputs, 1)
        self.activation = Sigmoid()

    # forward propagate input
    def forward(self, X):
        X = self.layer(X)
        X = self.activation(X)
        return X
```

- Los pesos de una capa dada también pueden ser inicializados después de que la capa sea definida en el constructor. Algunos ejemplos comunes son los esquemas de inicialización de pesos de Xavier y He. Por ejemplo:

```
...
xavier_uniform_(self.layer.weight)
```

Paso 3: Entrenar el modelo

- El proceso de entrenamiento requiere que se defina una función de pérdida y un algoritmo de optimización.
- Las funciones de pérdida más comunes son las siguientes
  - BCELoss: Pérdida de entropía cruzada binaria para la clasificación binaria.
  - CrossEntropyLoss: Pérdida de entropía cruzada categórica para la clasificación multiclase.
  - MSELoss: Pérdida media cuadrática para la regresión.
- El descenso de gradiente estocástico se utiliza para la optimización, y el algoritmo estándar lo proporciona la clase SGD, aunque hay otras versiones del algoritmo disponibles, como Adam.

```
# define the optimization
criterion = MSELoss()
optimizer = SGD(model.parameters(), lr=0.01, momentum=0.9)
```

- El entrenamiento del modelo implica la enumeración del DataLoader para el conjunto de datos de entrenamiento. En primer lugar, se requiere un bucle para el número de épocas de entrenamiento. A continuación, se requiere un bucle interno para los mini-lotes del descenso de gradiente estocástico.

```
...
# enumerate epochs
for epoch in range(100):
    # enumerate mini batches
    for i, (inputs, targets) in enumerate(train_dl):
        ...
```

- Cada actualización del modelo implica el mismo patrón general compuesto por:
  - Borrar el último gradiente de error.
  - Un pase hacia adelante de la entrada a través del modelo.
  - Cálculo de la pérdida para la salida del modelo.
  - Retropropagación del error a través del modelo.
  - Actualizar el modelo en un esfuerzo por reducir la pérdida.

```
...
# clear the gradients
optimizer.zero_grad()
# compute the model output
yhat = model(inputs)
# calculate loss
loss = criterion(yhat, targets)
# credit assignment
loss.backward()
# update model weights
optimizer.step()
```

Paso 4: Evaluar el modelo y Paso 5: Hacer predicciones

- Una vez ajustado el modelo, puede evaluarse en el conjunto de datos de prueba.

- Para ello, se utiliza el DataLoader para el conjunto de datos de prueba y se recogen las predicciones para el conjunto de prueba, comparando después las predicciones con los valores esperados del conjunto de prueba y calculando una métrica de rendimiento.

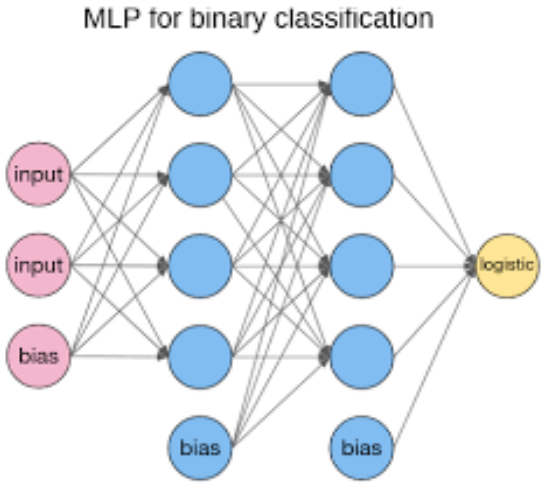
```
...
for i, (inputs, targets) in enumerate(test_dl):
    # evaluate the model on the test set
    yhat = model(inputs)
    ...
```

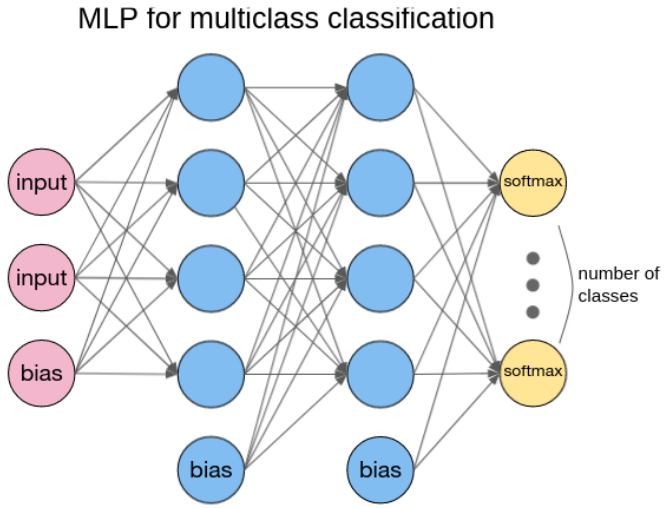
- Un modelo ajustado puede utilizarse para hacer una predicción sobre nuevos datos. Por ejemplo, podrías tener una sola imagen o una sola fila de datos y querer hacer una predicción. Esto requiere que envuelva los datos en una estructura de datos Tensor de PyTorch.
- Un tensor es la versión de PyTorch de un array de NumPy para contener datos. También permite realizar las tareas de diferenciación automática en el gráfico del modelo, como llamar a backward() cuando se entrena el modelo.
- La predicción también será un Tensor, aunque puedes recuperar el array NumPy separando el Tensor del gráfico de diferenciación automática y llamando a la función NumPy.

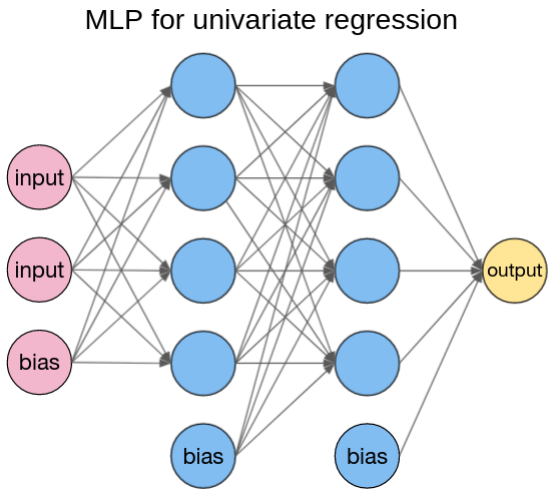
```
...
# convert row to data
row = Variable(Tensor([row]).float())
# make prediction
yhat = model(row)
# retrieve numpy array
yhat = yhat.detach().numpy()
```

Cómo desarrollar modelos de aprendizaje profundo en PyTorch

- En esta parte del módulo, descubrirás cómo desarrollar, evaluar y hacer predicciones con modelos de aprendizaje profundo estándar, incluidos los perceptrones multicapa (MLP) y las redes neuronales convolucionales (CNN).
- Un modelo de Perceptrón Multicapa, o MLP para abreviar, es un modelo de red neuronal estándar totalmente conectado. Está compuesto por capas de nodos en las que cada nodo está conectado a todas las salidas de la capa anterior y la salida de cada nodo está conectada a todas las entradas de los nodos de la capa siguiente.
- Un MLP es un modelo con una o más capas totalmente conectadas. Este modelo es apropiado para datos tabulares, es decir, datos tal y como aparecen en una tabla u hoja de cálculo con una columna para cada variable y una fila para cada variable. Hay tres problemas de modelado predictivo que puede querer explorar con un MLP; son la clasificación binaria, la clasificación multiclase y la regresión.
- Vamos a ajustar un modelo en un conjunto de datos real para cada uno de estos casos.
- Nota: Los modelos de esta sección son eficaces, pero no están optimizados.







MLP para clasificación binaria

- Utilizaremos el conjunto de datos de clasificación binaria (dos clases) de la Ionosfera para demostrar un MLP para la clasificación binaria. Este conjunto de datos consiste en predecir si hay una estructura en la atmósfera o no a partir de los retornos del radar.
- El conjunto de datos se descargará automáticamente usando Pandas, pero puedes aprender más sobre él aquí:
  - Conjunto de datos de la ionosfera (csv).
  - Descripción del conjunto de datos de la ionosfera.
- Utilizaremos un LabelEncoder para codificar las etiquetas de cadena a valores enteros 0 y 1. El modelo se entrenará con el 67% de los datos, y el 33% restante se utilizará para la evaluación, dividido utilizando la función train_test_split().
- Es una buena práctica utilizar la activación 'relu' con una inicialización de pesos 'He Uniform'. Esta combinación ayuda a superar el problema de los gradientes de fuga cuando se entrenan modelos de redes neuronales profundas.
- El modelo predice la probabilidad de la clase y utiliza la función de activación sigmoidea. El modelo se optimiza utilizando el descenso de gradiente estocástico y busca minimizar la pérdida de entropía cruzada binaria.
- Para ejecutar nuestro código: **python 1-mlp_binary_classification.py**

- Al ejecutar el ejemplo, primero se informa de la forma de los conjuntos de datos de entrenamiento y de prueba, luego se ajusta el modelo y se evalúa en el conjunto de datos de prueba. Por último, se realiza una predicción para una sola fila de datos.
- Nota: Sus resultados pueden variar debido a la naturaleza estocástica del algoritmo o del procedimiento de evaluación, o a las diferencias en la precisión numérica. Considere la posibilidad de ejecutar el ejemplo varias veces y compare el resultado medio.
- En este caso, podemos ver que el modelo logró una precisión de clasificación de alrededor del 94 por ciento y luego predijo una probabilidad de 0,99 de que la única fila de datos pertenezca a la clase 1.

MLP para clasificación multiclase

- Utilizaremos el conjunto de datos de clasificación multiclase de las flores del dataset IRIS para demostrar un MLP para la clasificación multiclase. Este problema consiste en predecir la especie de la flor del iris dadas las medidas de la flor.
- El conjunto de datos se descargará automáticamente usando Pandas, pero puedes aprender más sobre él aquí.
  - Conjunto de datos de Iris (csv).
  - Descripción del conjunto de datos de Iris.
- Dado que se trata de una clasificación multiclase, el modelo debe tener un nodo para cada clase en la capa de salida y utilizar la función de activación softmax. La función de pérdida es la entropía cruzada, que es adecuada para las etiquetas de clase codificadas con números enteros (por ejemplo, 0 para una clase, 1 para la siguiente, etc.).
- A continuación el script 2_mlp_multiclass_classification.py muestra el ejemplo completo de ajuste y evaluación de un MLP en el conjunto de datos de flores IRIS.

- Al ejecutar el ejemplo, primero se informa de la forma de los conjuntos de datos de entrenamiento y de prueba, luego se ajusta el modelo y se evalúa en el conjunto de datos de prueba. Por último, se realiza una predicción para una sola fila de datos.
- Nota: Sus resultados pueden variar debido a la naturaleza estocástica del algoritmo o del procedimiento de evaluación, o a las diferencias en la precisión numérica. Considere la posibilidad de ejecutar el ejemplo varias veces y compare el resultado medio.
- En este caso, podemos ver que el modelo logró una precisión de clasificación de alrededor del 98 por ciento y luego predijo la probabilidad de que una fila de datos perteneciera a cada clase, aunque la clase 0 tiene la mayor probabilidad.

MLP para regresión

- Utilizaremos el conjunto de datos de regresión de viviendas de Boston para demostrar un MLP para el modelado predictivo de regresión. Este problema consiste en predecir el valor de la vivienda basándose en las propiedades de la misma y del vecindario. El conjunto de datos se descargará automáticamente usando Pandas, pero puedes aprender más sobre él aquí.
  - Conjunto de datos de viviendas de Boston (csv).
  - Descripción del conjunto de datos de viviendas de Boston.
- Este es un problema de regresión que implica la predicción de un único valor numérico. Como tal, la capa de salida tiene un solo nodo y utiliza la función de activación por defecto o lineal (sin función de activación). La pérdida del error medio cuadrático (mse) se minimiza al ajustar el modelo.
- Recordemos que esto es una regresión, no una clasificación; por lo tanto, no podemos calcular la precisión de la clasificación. El ejemplo completo de ajuste y evaluación de un MLP en el conjunto de datos de viviendas de Boston se muestra en el script:

3_mlp_regression.py

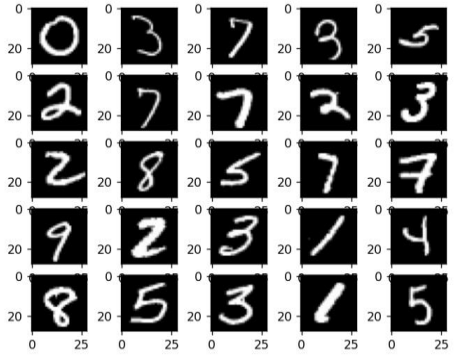
- Al ejecutar el ejemplo, primero se informa de la forma de los conjuntos de datos de entrenamiento y de prueba, luego se ajusta el modelo y se evalúa en el conjunto de datos de prueba. Por último, se realiza una predicción para una sola fila de datos.
- En este caso, podemos ver que el modelo alcanzó un MSE de aproximadamente 82, lo que supone un RMSE de aproximadamente nueve (las unidades son miles de dólares). A continuación, se predice un valor de 21 para el ejemplo único.

CNN para clasificación de imágenes

- Las redes neuronales convolucionales, o CNN, son un tipo de red diseñada para la entrada de imágenes. Se componen de modelos con capas convolucionales que extraen características (llamadas mapas de características) y capas de agrupación que destilan las características hasta los elementos más destacados.
- Las CNN son las más adecuadas para las tareas de clasificación de imágenes, aunque pueden utilizarse en una amplia gama de tareas que toman imágenes como entrada. Una tarea popular de clasificación de imágenes es la clasificación de dígitos manuscritos del MNIST. Se trata de decenas de miles de dígitos escritos a mano que deben clasificarse como un número entre 0 y 9.
- La API de torchvision proporciona una función conveniente para descargar y cargar este conjunto de datos directamente.
- El primer script: "4_show_mnist.py" carga el conjunto de datos y traza las primeras imágenes.
- Podemos entrenar un modelo CNN para clasificar las imágenes del conjunto de datos MNIST. Tenga en cuenta que las imágenes son matrices de datos de píxeles en escala de grises, por lo tanto, debemos añadir una dimensión de canal a los datos antes de que podamos utilizar las imágenes como entrada al modelo.
- Es una buena idea escalar los valores de los píxeles desde el rango por defecto de 0-255 para tener una media cero y una desviación estándar de 1. Para más información sobre el escalado de los valores de los píxeles, vea el tutorial
- El ejemplo completo de ajuste y evaluación de un modelo CNN en el conjunto de datos MNIST se muestra en el script:

5_cnn_mnist.py

- La ejecución informa primero de la forma de los conjuntos de datos de entrenamiento y de prueba, y luego ajusta el modelo y lo evalúa en el conjunto de datos de prueba. En este caso, podemos ver que el modelo alcanzó una precisión de clasificación de aproximadamente el 98% en el conjunto de datos de prueba. Podemos ver que el modelo predijo la clase 5 para la primera imagen del conjunto de entrenamiento.



Conclusiones

- Se han repasado las fases principales de un proyecto de IA
- Se ha presentado el framework de programación de Deep Learning Pytorch y sus características frente a otros.
- Hemos instalado Pytorch en nuestro entorno Python.
- Hemos enlazado las fases de un proyecto de IA con la programación.
- Se han presentado diferentes ejemplos de redes neuronales que resuelven diferentes problemas con distintos datasets. En primer lugar un MLP para clasificación binaria, en segundo lugar un MLP para clasificación multiclase, a continuación un MLP para regresión y finalmente una CNN para clasificación de imágenes.

Universidad Politécnica de Madrid

Herramientas software para la realización de proyectos de AI aplicados a la investigación

Proyecto de IA basado en Pytorch

Alberto Belmonte Hernández