Introduction to Deep Learning & Neural Networks with keras

Week 3

Deep Learning Libraries

1. Deep Learning Libraries

En este apartado se hace un breve resumen de las librerías más importantes en deep learning.

- TensorFlow
- keras
- PyTorch
- Theano: esta última no entraremos en detalle ya que está deprecada y no ofrece soporte.

TensorFlow es la librería más popular y principalmente se utiliza en la producción de modelos de deep learning. Tiene una comunidad muy grande. Fue desarrollado por Google y lanzado al público en 2015, a día de hoy Google la sigue utilizando en temas de investigación.

PyTorch, es framework de Torch, y admite algoritmos de aprendizaje automático que se ejecutan en GPU en particular.
Fue lanzado en 2016 y ha ganado un gran interés últimamente y se está convirtiendo en el idioma preferido sobre TensorFlow, especialmente en entornos de investigación académica y aplicaciones de deep learning que requieren la optimización de expresiones personalizadas. PyTorch es compatible y se utiliza activamente en Facebook.

Sin embargo, tanto PyTorch como TensorFlow no son fáciles de usar, y tienen una curva de aprendizaje empinada. Así que para las personas que están empezando a aprender deep learning, la mejor opción es empezar por la librería de Keras.

Keras es una API de alto nivel para crear modelos de deep learning. Es muy fácil de utilizar y tiene una sintásis muy sencilla. Keras tiene a bajo nivel código de TensorFlow

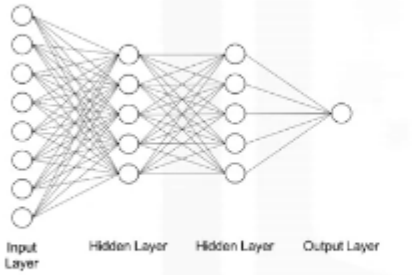
En resumen:

- Para hacer un proyecto de deep learning de forma rápida, lo mejor es utilizar Keras.
- Para tener más control sobre los diferentes nodos y capas en la red, así como analizar su comportamiento en un período largo de tiempo, lo mejor es utilizar PyTorch o TensorFlow.

Regression with Keras

2. Regression Models with keras

En este punto se explica un ejemplo básico de como crear una red neuronal con keras. En la imagen adjunta se pueden ver los pasos que se siguen el código para crear la red.




```
import keras
from keras.models import Sequential
from keras.layers import Dense

model = Sequential()

n_cols = concrete_data.shape[1]

model.add(Dense(5, activation='relu', input_shape=(n_cols,)))
model.add(Dense(5, activation='relu'))
model.add(Dense(1))

model.compile(optimizer='adam', loss='mean_squared_error')
model.fit(predictors, target)

predictions = model.predict(test_data)
```

1. Dividir los datos en predicciones y target.
   Si nos fijamos en la imagen debido a que nuestra red consiste en una pila lineal de capas, entonces usamos el modelo secuencial. Hay dos modelos en la biblioteca de Keras:

- Secuencial
- Modelo utilizado con la API funcional.

Una vez creado el modelo, se construyen las capas. Para eso, necesitaríamos importar el tipo de capas «Densa» de «keras.layers». (Densa porque todos los nodos están conectados con todos)
Luego usamos el método add para agregar cada capa densa. Se especifica el número de neuronas en cada capa y la función de activación que queremos utilizar (ReLU es una de las funciones de activación recomendadas para capas ocultas)

Y para la primera capa oculta necesitamos pasar el parámetro «input_shape», que es el número de columnas o predictores en nuestro conjunto de datos. Luego repetimos lo mismo para la otra capa oculta, por supuesto, sin el parámetro input_shape, y creamos nuestra capa de salida con un nodo.

Ahora, para el entrenamiento, necesitamos definir un optimizador y la métrica de error. Para este ejemplo, se utiliza el descenso de gradiente como nuestro algoritmo de minimización o optimización, y el error cuadrado medio como nuestra medida de pérdida entre el valor de previsto y el ground truth.

Otros algoritmos más eficientes que el descenso de gradiente para aplicaciones de deep learning son entre otros «Adam». Una de las principales ventajas del optimizador «adam» es que no necesitas especificar la tasa de aprendizaje. Así que esto nos ahorra la tarea de optimizar la tasa de aprendizaje para nuestro modelo.

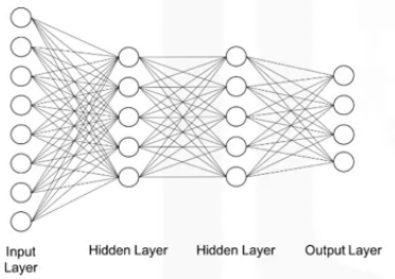
Luego usamos el método fit para entrenar nuestro modelo. Una vez completado el entrenamiento, podemos empezar a hacer predicciones usando el método de predicción.

Classification with Keras

3. Classification Models with Keras

En este punto, vamos a aprender cómo usar la biblioteca de Keras para crear modelos para problemas de clasificación. Para ello, se explica un ejemplo, en el que un modelo de predicción nos indica lo bueno o no, que es comprar un cierto modelo de

coche, según su precio, comodidad, número de personas que pueden viajar y mantenimiento.




```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.utils import to_categorical

model = Sequential()

n_cols = car_data.shape[1]

target = to_categorical(target)

model.add(Dense(5, activation='relu', input_shape=(n_cols,)))
model.add(Dense(5, activation='relu'))
model.add(Dense(4, activation='softmax'))

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

En la imagen superior podemos ver el resumen del código, en el que se aplica el modelo de clasificación con keras. A tener en cuenta:

- Para problemas de clasificación, no podemos usar la columna de destino tal cual; en realidad necesitamos transformar la columna en una matriz con valores binarios similares a la codificación One Hot como la salida que se muestra aquí. Lo conseguimos fácilmente usando la función «to_categorical» del paquete de utilidades de Keras. En otras palabras, nuestro modelo en lugar de de tener sólo una neurona en la capa de salida, tendría cuatro neuronas, ya que nuestra variable objetivo consta de cuatro categorías.

- También especificamos la función softmax como la función de activación para la capa de salida, de modo que la suma de los valores pronosticados de todas las neuronas en la capa de salida sume muy bien a 1.

- La salida del método de predicción de Keras sería algo así como lo que se muestra aquí. Para cada punto de datos, la salida es la probabilidad de que la decisión de comprar un coche dado pertenece a una de las cuatro clases. Para cada punto de datos, las probabilidades deben sumar a 1, y cuanto mayor sea la probabilidad, más seguro es el algoritmo de que un punto de datos pertenece a la clase respectiva. Así que para el primer punto de datos o el primer coche en el conjunto de pruebas, la decisión sería 0 que significa no aceptable, ya que la primera probabilidad es la más alta, con un valor de 0.99 o cerca de 1, en este caso.

- Observe cómo las probabilidades para la decisión 0 y la decisión 1 están muy cerca. Por lo tanto, el modelo no es muy seguro, pero se inclinaría hacia la aceptación de la compra de estos coches.

```
model.predict(test_data)
```

```
array([[9.9978679e-01, 2.1028408e-04, 1.0243932e-06, 1.8633460e-06],
       [9.9978679e-01, 2.1028408e-04, 1.0243932e-06, 1.8633460e-06],
       [9.9978679e-01, 2.1028408e-04, 1.0243932e-06, 1.8633460e-06],
       ...,
       [4.9434581e-01, 4.9560347e-01, 4.4486104e-03, 5.6021004e-03],
       [4.9434581e-01, 4.9560347e-01, 4.4486104e-03, 5.6021004e-03],
       [4.9434581e-01, 4.9560347e-01, 4.4486104e-03, 5.6021004e-03]],
      dtype=float32)
```