Redes neuronales en Tensorflow

Redes neuronales

- Una ANN tiene cuatro componentes principales:
  1. Capas, donde ocurre el aprendizaje (capa de entrada, ocultas y de salida).
  2. Datos de entrada y salida:
     - Datos para el entrenamiento/aprendizaje (entrada);
     - Predicción o etiqueta (salida)
  3. Función de pérdida: métrica utilizada para estimar el rendimiento de la fase de aprendizaje
  4. Optimizador: método para mejorar el aprendizaje, mediante la actualización del conocimiento en la red (pesos para la ponderación)

La ANN toma los datos de entrada para alimentar al conjunto de capas.
El rendimiento se evalúa con la función de pérdida.
El conocimiento se mejora a través del optimizador.

Ciclo de un modelo utilizando ANN

1. Definir el modelo
2. Compilar el modelo
3. Ajustar el modelo
4. Evaluar el modelo
5. Realizar predicciones/clasificaciones/generaciones

Definir el modelo

- Implica seleccionar el tipo de modelo (MLP, RNN, CNN, etc.)
- Luego se debe determinar la arquitectura de la ANN a utilizar.
- Por último, deben definirse las capas del modelo y configurarlas, especificando el número de neuronas, la función de activación y las conexiones entre las capas del modelo.

Compilar el modelo

- Requiere definir una función de pérdida (loss function) para el entrenamiento (optimización de los pesos de la ANN):
  - Mean squared error, cross-entropy, etc.
- Requiere seleccionar un algoritmo para la optimización de los pesos:
  - Stochastic gradient descent (SGD).
  - Variantes de SGD: Adam, etc.
  - Otros
- Requiere determinar métricas relevantes para monitorear el desempeño de la etapa de entrenamiento.

Ajustar el modelo

- Requiere seleccionar la configuración de entrenamiento:
  - Número de épocas (epochs): ciclos a realizar sobre el conjunto de entrenamiento)
  - Tamaño del batch (batch size): número de muestras que se utilizan en una iteración para estimar el error del modelo.
- El entrenamiento aplica el algoritmo de optimización especificado en la etapa previa para minimizar la función de pérdida seleccionada y actualiza los pesos del modelo utilizando un método de propagación hacia atrás (backpropagation) del error.
- Es la etapa que consume mayor tiempo de cómputo, dependiendo de la complejidad del problema, la complejidad del modelo, el hardware utilizado y el tamaño del conjunto de entrenamiento.

Evaluar el modelo

- Requiere determinar un conjunto de datos de validación.
- Datos no utilizados en la etapa de entrenamiento, para evitar sesgos.
- El tiempo requerido para la evaluación es proporcional al tamaño del conjunto de datos de validación (esta etapa es mucho más rápida que el entrenamiento).

Predecir/clasificar/generar

- Validar el modelo sobre datos desconocidos.
  - Predicción: datos para los cuales no se conoce el valor a predecir.
  - Clasificación: datos para los cuales no se conoce la clase a la que pertenecen.
  - Generación: Crear nuevas imágenes, datos tabulares, series temporales, etc.

- Los modelos pueden salvarse para ser utilizados posteriormente.
- Un modelo previamente creado (y entrenado) puede ser reentrenado/recalibrado sobre otros conjuntos de datos.

Tensorflow

- Biblioteca de código abierto desarrollada por Google.
- Diseñada para el cálculo eficiente de flujos de grafos, es muy útil para tareas de deep learning tasks.
- Ejecuta en una o multiples CPUs/GPUs.
- La API de Tensorflow (a partir de la versión 2) es Keras

Instalación de Tensorflow

- Python 3.6 o superior o conda/anaconda
- La manera más simple es instalar TensorFlow con pip.

```
sudo pip install tensorflow
```

- No es necesario configurar GPU inicialmente.
- Para confirmar que TensorFlow está instalado:
  - Crear un archivo versions.py.

    ```
    import tensorflow
    print(tensorflow.__version__)
    ```

```
python versions.py
```

- Salida: 2.2.0, confirma que TensorFlow está correctamente instalado

Instalación de Tensorflow

- Algunos mensajes comunes (warnings)

```
Your CPU supports instructions that this TensorFlow binary was
not compiled to use: AVX2 FMA

XLA service 0x7fde3f2e6180 executing computations on platform
Host. Devices:

StreamExecutor device (0): Host, Default Version
```

- No impiden la ejecución de código

La biblioteca Keras

Keras

- Keras: biblioteca open source para deep learning en Python
- Permite simplificar el modelado y el desarrollo de modelos de aprendizaje en Python
- Fue diseñada para proporcionar simplicidad a los desarrollos sobre entornos existentes (TensorFlow, Theano, PyTorch), que están más enfocados en la eficiencia, flexibilidad, investigación y aplicabilidad
- Keras proporciona una API simple y clara para definir modelos de deep learning y su posterior evaluación, en códigos compactos de pocas líneas.
- Keras permite utilizar los entornos y bibliotecas de deep learning más difundidas (TensorFlow, Theano, CNTK) como backend para realizar los cálculos, aprovechando su eficiencia, uso de GPUs, etc.

Keras en TensorFlow 2

- TensorFlow 2 (2019) integra Keras como API y la sugiere como interfaz para el Desarrollo (interfaz tf.keras)
  - Standalone Keras: Proyecto open source genérico que da soporte a TensorFlow, Theano y CNTK como backends.
  - tf.keras: Keras como API integrada en TensorFlow 2.
- tf.keras permite utilizar una única biblioteca, en lugar de dos separadas.

```
import tensorflow as tf
model = tf.keras.Sequential()
```

Ciclo de un modelo utilizando ANN en Keras

1. Definir el modelo
2. Compilar el modelo
3. Ajustar el modelo
4. Evaluar el modelo
5. Realizar predicciones/clasificaciones/generaciones

Keras: definir el modelo

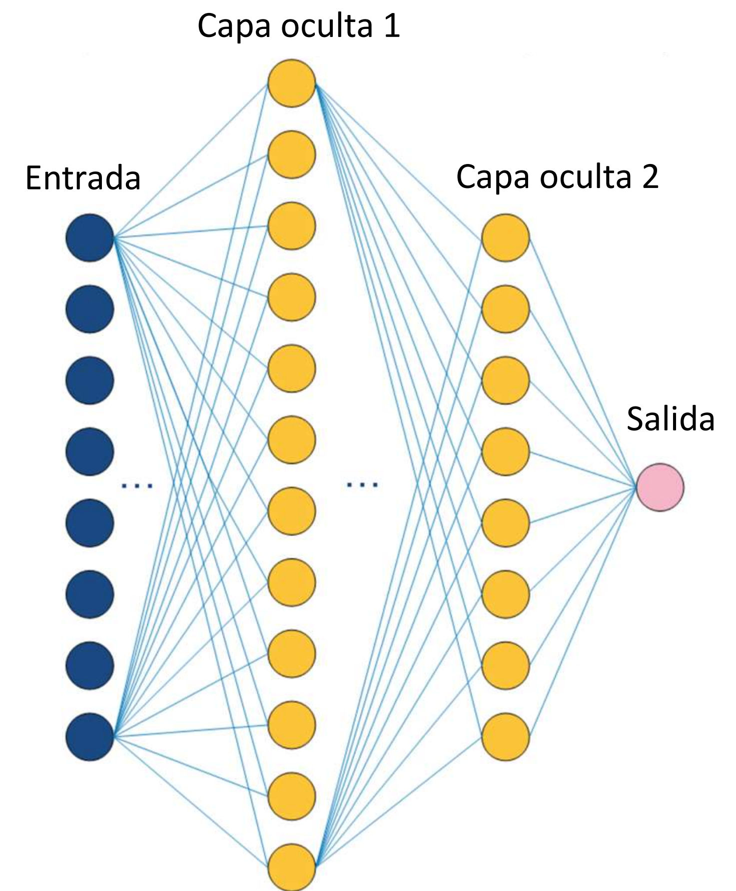
- Implica seleccionar el tipo de modelo (MLP, RNN, CNN, etc.)
- Luego se debe determinar la arquitectura de la ANN a utilizar.
- Por ultimo, deben definirse las capas del modelo y configurarlas, especificando el número de neuronas, la función de activación y las conexiones entre las capas del modelo.

- En Keras:
- Dos opciones:
  - API secuencial: sintaxis estándar, más simple, `model = Sequential()`
  - API funcional: más flexible.

Keras: definir el modelo con la API secuencial

- Implica definir una clase Sequential y especificar secuencialmente las capas del modelo, desde la entrada a la salida.
- Ejemplo: modelo con dos capas densas de 12 y 8 neuronas respectivamente

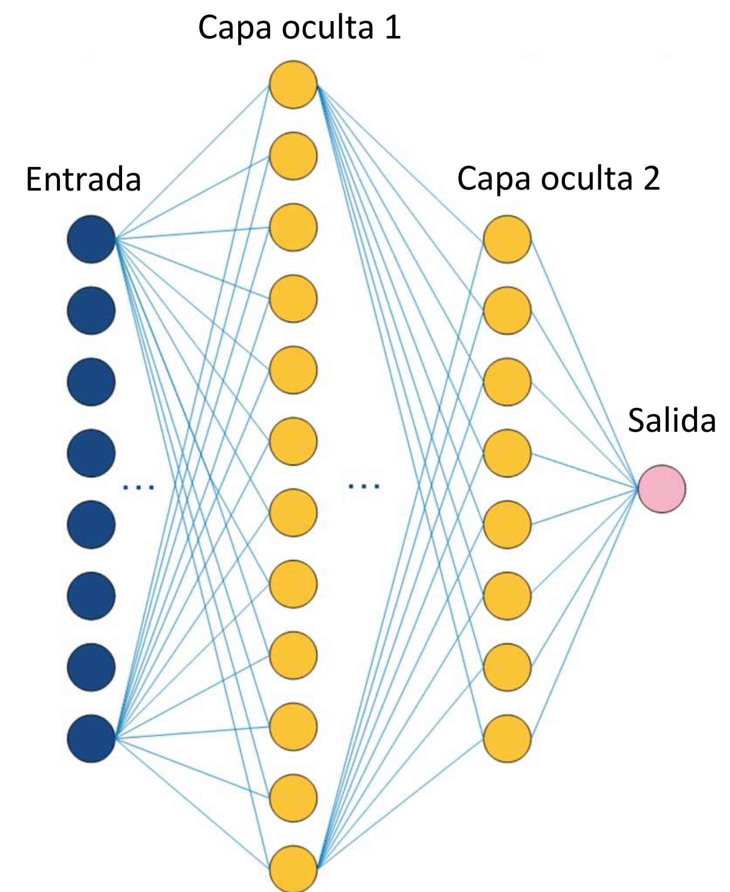
```
model = Sequential([Dense(12,
input_dim=8), Activation('relu’),
Dense(8), Activation('softmax'),
Dense(1), Activation('sigmoid')])
```



Keras: definir el modelo con la API secuencial

- Implica definir una clase Sequential y especificar secuencialmente las capas del modelo, desde la entrada a la salida.
- De forma equivalente, se puede utilizar el método add() para agregar capas

```
model = Sequential()
model.add(Dense(12, input_dim=8,
init='uniform', activation='relu'))
model.add(Dense(8, init='uniform',
activation='relu'))
model.add(Dense(1, init='uniform',
activation='sigmoid'))
```



Keras: definir el modelo con la API secuencial

- Ejemplo (2): modelo MLP secuencial que acepta seis entradas, tiene una capa oculta con 10 neuronas y una capa de salida con una neurona para predecir un valor numérico

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
# definir el modelo
model = Sequential()
model.add(Dense(10, input_shape=(6,)))
model.add(Activation('relu'))
model.add(Dense(1))
model.add(Activation('tanh'))
```

- La capa de entrada se define al inicio, con el argumento `input_shape`. En el ejemplo, el modelo espera recibir como entrada un vector de dimensión 6.

Keras: definir el modelo con la API secuencial

- La API secuencial es simple de utilizar, ya que implica invocar `model.add()` hasta agregar todas las capas del modelo.
- Ejemplo: deep MLP con cinco capas ocultas.

```
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
# define the model
model = Sequential()
model.add(Dense(100, input_shape=(8,)))
model.add(Dense(80), activation='relu')
model.add(Dense(30), activation='relu')
model.add(Dense(10), activation='relu')
model.add(Dense(5), activation='relu')
model.add(Dense(1), activation='sigmoid')
```

Keras: definir el modelo con la API funcional

- API más compleja, pero también más flexible.
- Implica conectar de manera explícita la salida de una capa con la entrada de otra capa. Debe especificarse cada conexión.
- El primer paso es definir la capa de entrada, con la clase `Input` e indicar su dimensión.
- Debe mantenerse una referencia a la capa de entrada al definir el modelo

```
x_in = Input(shape=(8,))
```

- Luego se conecta una capa totalmente conectada a la entrada, creando una nueva capa (oculta) y pasando como argumento la capa previa (entrada).
- El proceso retorna una referencia a la salida de la nueva capa.

```
x = Dense(10)(x_in)
```

Keras: definir el modelo con la API funcional

- La salida de la capa oculta se conecta a una nueva capa o a una capa de salida

```
x_out = Dense(1)(x)
```

- Luego de definir las capas y las conexiones, se define un objeto `Model` y se especifican las capas de entrada y salida.

```
from tensorflow.keras import Model
from tensorflow.keras import Input
from tensorflow.keras.layers import Dense
# define the layers
x_in = Input(shape=(8,))
x = Dense(10)(x_in)
x_out = Dense(1)(x)
# define the model
model = Model(inputs=x_in, outputs=x_out)
```

Keras: definir el modelo con la API funcional

- La API funcional es más flexible y permite definir diseños para modelos complejos.
- Modelos con múltiples entradas (vectores separados)
- Modelos con múltiples salidas (por ejemplo, un número y una palabra).

Keras: compilar el modelo

- Requiere definir una función de pérdida (loss function) para el entrenamiento (optimización de los pesos de la ANN).
- Requiere seleccionar un algoritmo para la optimización de los pesos:
- Requiere determinar métricas relevantes para monitorear el desempeño de la etapa de entrenamiento.

- En Keras: invocar una función para compilar el modelo con la configuración seleccionada.
- La compilación preparara las estructuras de datos requeridas para el uso eficiente del modelo definido.

Keras: compilar el modelo

- El optimizador se indica como un string de una clase provista (ejemplo: ‘sgd‘ para stochastic gradient descent)

```
model.compile(optimizer='sgd', loss='mse’)
```

- También puede configurarse una instancia de un optimizador y utilizarse

```
opt = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=opt, loss='binary_crossentropy’)
```

- Los optimizadores de Keras se listan con `tf.keras Optimizers`
- Las funciones de pérdida de Keras se listan con `tf.keras Loss Functions`, las más utilizadas son:
  - ‘binary_crossentropy‘ para problemas de clasificación binaria.
  - ‘sparse_categorical_crossentropy‘ para problemas de clasificación multiclase.
  - ‘mse‘ (mean squared error) para problemas de regresión.

Keras: ajustar el modelo

- Requiere seleccionar la configuración de entrenamiento
  - Número de épocas: ciclos a realizar sobre el conjunto de entrenamiento)
  - Tamaño del batch (batch size): número de muestras que se utilizan en una época para estimar el error del modelo.
- Aplica el algoritmo de optimización para minimizar la function de pérdida y actualiza los pesos del por backpropagation.
- En Keras, implica invocar a una función para realizar el entrenamiento. La función solo retorna el control una vez que el entrenamiento finaliza.

```
model.fit(X, y, epochs=100, batch_size=32)
```

Keras: ajustar el modelo

- Durante el entrenamiento se visualiza el progreso para cada época (o conjunto de épocas).
- El parámetro ‘verbose’ permite controlar el nivel de detalle
  - verbose=2, muestra salidas [resumidas] para cada época
  - verbose=0, deshabilita reportes

```
model.fit(X, y, epochs=100, batch_size=32, verbose=0)
```

Keras: evaluar el modelo

- Requiere determinar un conjunto de datos de validación.
- Datos no utilizados en la etapa de entrenamiento, para evitar sesgos.
- El tiempo requerido para la evaluación es proporcional al tamaño del conjunto de datos de validación (mucho más rápido que el entrenamiento)

- En Keras, implica invocar a una función con el conjunto de datos de validación y obtener un valor de la función de pérdida [y otras métricas relevantes].

```
loss = model.evaluate(X, y, verbose=0)
```

Keras: predecir/clasificar/generar

- Validar el modelo sobre datos desconocidos
- Predicción: datos para los cuales no se conoce el valor a predecir.
- Clasificación: datos para los cuales no se conoce la clase a la que pertenecen.
- Generación: Crear nuevas imágenes, datos tabulares, series temporales, etc.

```
yhat = model.predict(X)
```

La biblioteca scikit-learn

Scikit-learn

- Biblioteca open source de Python para aprendizaje automático.
- Provee implementaciones de múltiples algoritmos de clasificación, regresión, agrupamiento y otras técnicas.
  - Máquinas de soporte vectorial (SVM), Random Forest, Gradient boosting, K-means, DBSCAN, redes neuronales.
- Diseñada para interoperar con las bibliotecas numéricas y científicas NumPy y SciPy.
- Provee muchas funciones para manejo de datos, reportes y estadísticas, etc.

Ejemplo: MLP para clasificación binaria

- Dataset: Ionosphere, datos de clasificación binaria (dos clases).
- Indica si una estructura [determinada por lecturas de radar] está en la atmósfera o no.
- El dataset se carga automáticamente utilizando la biblioteca Pandas.

- Se diseña un modelo que predice la clase, utilizando la función de activación sigmoide.
- El modelo se entrena con la variante Adam de SGD, minimizando la entropía cruzada.
- Código disponible en `github.com/nesmachnow/Curso-GANs`

Ejemplo: MLP para clasificación binaria

```
from pandas import read_csv
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from tensorflow.keras import Sequential
from tensorflow.keras.layers import Dense
# Cargar el dataset
df = read_csv('https://raw.githubusercontent.com/
      jbrownlee/Datasets/master/ionosphere.csv', header=None)
# Particionar en columnas de entrada y de salida
X, y = df.values[:, :-1], df.values[:, -1]
# Verificar que los datos son números de punto flotante
X = X.astype('float32')
# Codificar strings a enteros
y = LabelEncoder().fit_transform(y)
```

Codifica etiquetas (strings) a enteros (0/1)

Ejemplo: MLP para clasificación binaria

```
# Particionar en datasets de entrenamiento y validación
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.33)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
# Determinar el número de características de entrada
n_features = X_train.shape[1]
# Definir el modelo
model = Sequential()
model.add(Dense(10, activation='relu', kernel_initializer='he_normal',
input_shape=(n_features,)))
model.add(Dense(8, activation='relu', kernel_initializer='he_normal'))
model.add(Dense(1, activation='sigmoid'))
# Compilar el modelo
model.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])
```

Se entrena sobre 2/3 del dataset

ReLU + he_normal: robusto para evitar problemas de gradiente nulo

Ejemplo: MLP para clasificación binaria

```
# Ajustar el modelo
model.fit(X_train, y_train, epochs=150, batch_size=32, verbose=0)
# Evaluar el modelo
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print('Test Accuracy: %.3f' % acc)
# Utilizar el modelo para hacer una predicción
row = [1,0,0.99539,-0.05889,0.85243,0.02306,0.83398,-
0.37708,1,0.03760,0.85243,-0.17755,0.59755,-0.44945,0.60536,-
0.38223,0.84356,-0.38542,0.58212,-0.32192,0.56971,-0.29674,0.36946,-
0.47357,0.56811,-0.51171,0.41078,-0.46168,0.21266,-0.34090,0.42267,-
0.54487,0.18641,-0.45300]
yhat = model.predict([row])
print('Predicted: %.3f' % yhat)
```

Ejemplo (2): clasificación de dígitos manuscritos

- Dataset MNIST: 60000 imágenes de 28×28 pixels de digitos manuscritos, en escala de grises.
- Dataset estándar para aprendizaje.
- Código disponible en github.com/nesmachnow/Curso-GANs

Ejemplo (2): clasificación de dígitos manuscritos

```
import tensorflow as tf
mnist = tf.keras.datasets.mnist
(x_train, y_train), (x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

Carga y prepara el conjunto de datos MNIST

```
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(input_shape=(28, 28)),
  tf.keras.layers.Dense(128, activation='relu'),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation='softmax')
])
```

Construye un modelo apilando capas.

```
model.compile(optimizer='adam',
  loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test,  y_test, verbose=2)
```

Entrena y evalúa el modelo

Ejemplo (2): clasificación de dígitos manuscritos

```
Epoch 1/5
1875/1875 [=================] - 3s 2ms/step - loss: 0.2935 - accuracy: 0.9155
Epoch 2/5
1875/1875 [=================] - 3s 2ms/step - loss: 0.1430 - accuracy: 0.9573
Epoch 3/5
1875/1875 [=================] - 3s 2ms/step - loss: 0.1073 - accuracy: 0.9670
Epoch 4/5
1875/1875 [=================] - 3s 2ms/step - loss: 0.0880 - accuracy: 0.9725
Epoch 5/5
1875/1875 [=================] - 3s 2ms/step - loss: 0.0745 - accuracy: 0.9761

313/313 - 0s - loss: 0.0707 - accuracy: 0.9785
[0.07073566317558289, 0.9785000085830688]
```