APPSA - Práctica 2: Reconocimiento de Locutor en VoxCeleb

Master en Ciencia de Datos – Escuela Politécnica Superior,

Universidad Autónoma de Madrid

Guillermo Hoyo Bravo

Abril, 2022

Objetivo

Explorar y evaluar un sistema de Reconocimiento de Locutor basado en embeddings (x-vectors) sobre la tarea de VoxCeleb

2. Identificación de líneas claves de código

Ya que como se ha mencionado previamente, el script **trainSpeakerNet.py** es el principal, vamos a explorar el código de dicho script para entender el funcionamiento del sistema.

PREGUNTAS: Incluye las respuestas a estas preguntas en la memoria de la práctica:

- **¿Qué línea de código se corresponde con el entrenamiento del modelo?**

Línea 169: #Entrenar Red/Modelo

loss, traineer = s.train_network(loader=trainLoader);

En esta línea obtenemos el entrenamiento del modelo. Devuelve una variable loss con los valores de la función y otra varaible traineer que contiene los valores de error de la fase de entrenamiento.

- **¿Qué comando carga un modelo previamente entrenado?**

Líneas 101 a 125: #Cargar Modelo (101- 110), #Cargar pesos del modelo (111 - 125)

Líneas [101- 110] - Cargar Modelo: Aquí se crea el modelo a entrenar.

s = SpeakerNet(**vars(args));

Líneas [111 - 125] - Cargar pesos del modelo preentrenado:

Aquí se cargan los Pesos necesarios para el modelo, y se cargan los parámetros previamente entrenados. Depende de si el modelo inicial tiene argumentos o no, se utilizará una de estas dos opciones:

Línea 116:

s.loadParameters(modelfiles[-1]);

Línea 119:

s.loadParameters(args.initial_model);

Línea 112: Carga los valores del modelo

modelfiles = glob.glob('%s/model0*.model'%model_save_path)

- **¿Qué código evalúa el rendimiento de la red neuronal?**

Linea 129:

sc, lab, trials = s.evaluateFromList(args.test_list, print_interval=100, test_path=args.test_path, eval_frames=args.eval_frames)

Línea 176 - Validación desde test_list:

sc, lab, _ = s.evaluateFromList(args.test_list, print_interval=100, test_path=args.test_path, eval_frames=args.eval_frames)

Return:

Sc = all_scores

Lab = all_labels

_ = all_trials

La diferencia entre ambas lineas mencionadas es que en la segunda solo entra una vez cada varias epocas, definidas por el argumentotest_interval.

Explicar las 2

- **¿Qué variable contiene las puntuaciones (scores) de la lista de trials? ¿En qué fichero se guardarán dichas puntuaciones?**

La variable sc es la que define y gurda los scores, mencionada en la pregunta anterior.

Los scores se guardan en el fichero definido en la línea 135:

userinp = result_save_path + '/scores_test.txt'#input()

- **¿Qué argumento controla el número de épocas (iteraciones) que se entrena el modelo?**

max_epoch: Es el argumento que controla el número total de épocas.

Mientras que la variable it es la iteración o época que está sucediendo en el momento (bucle)

- **¿Qué parámetro determina la función de coste?**

Línea 29 - Parámetro --trainfunc:

parser.add_argument('--trainfunc', type=str, default="", help='Loss function');

- **¿Para qué se usa el parámetro test_interval?**

Línea 27 – Número de épocas antes de evaluación(test):

parser.add_argument('--test_interval', type=int, default=10, help='Test and save every [test_interval] epochs');

- **¿Qué argumento necesitarías modificar para cambiar el tipo (arquitectura) del modelo que quieres entrenar?**

Línea 61 - Parametro --model:

parser.add_argument('--model', type=str, default="", help='Name of model definition');

- **¿Para qué se utiliza el parámetro nClasses? ¿Cómo obtendrías este valor a partir de tu conjunto de datos? Escribe el comando que utilizarías (por ejemplo, en Linux).**

Línea 44 – Número de Clases/emisores:

parser.add_argument('--nClasses', type=int, default=5994, help='Number of speakers in the softmax layer, only for softmax-based losses');

El número de hablantes no se encuentra en ningún lado, pero en la práctica se explican los ficheros que se van a utilizar para entrenamiento (/data/voxceleb2), y para evaluación(/data/voxceleb1).

Dentro de cualquiera de estas carpetas se encuentran más subcarpetas, una por emisor con sus diferentes .wav, por lo que contando este número de carpetas podemos obtenerlo de dos maneras:

Desplazándonos a los directorios mencionados y ejecutando el comando:

```
!wc -l
```

O desde donde nos encontramos encontrar el archivo buscado y con una tubería concatenar el comando recién mencionado:

```
!ls 'data/voxceleb1' | wc -l
```

Resultado: 40

```
!ls 'data/voxceleb2' | wc -l
```

Resultado: 4953

3. Evaluación de modelos preentrenados

En el directorio pretrained_models/ podemos encontrar dos modelos previamente entrenados con la base de datos VoxCeleb2 dev (partición de desarrollo).

Para evaluar cada uno de estos modelos previamente entrenados, podemos utilizar los siguientes comandos:

First Model:

```
!python ./trainSpeakerNet.py --eval --model ResNetSE34L --log_input True \
      --trainfunc angleproto --save_path exps/test_lite --eval_frames 400 \
      --test_list lists/veri_test.txt \
      --initial_model pretrained_models/baseline_lite_ap.model
```

```
Embedding size is 512, encoder SAP.
Initialised AngleProto
Initialised Adam optimizer
Initialised step LR scheduler
Model has 1437080 parameters
Model pretrained_models/baseline_lite_ap.model loaded!
Reading 4700 of 4715: 28.56 Hz, embedding size 512
Computing 37700 of 37720: 2068.46 Hz
EER 2.1792
```

Second Model

```
!python ./trainSpeakerNet.py --eval --model ResNetSE34V2 --log_input True \
  --encoder_type ASP --n_mels 64 --trainfunc softmaxproto \
  --save_path exps/test_v2 --eval_frames 400 --test_list lists/veri_test.txt \
  --initial_model pretrained_models/baseline_v2_ap.model
```

```
Embedding size is 512, encoder ASP.
Initialised Softmax Loss
Initialised AngleProto
Initialised SoftmaxPrototypical Loss
Initialised Adam optimizer
```

Initialised step LR scheduler

Model has 11103416 parameters

Model pretrained_models/baseline_v2_ap.model loaded!

Reading 4700 of 4715: 4.24 Hz, embedding size 512

Computing 37700 of 37720: 2040.67 Hz

EER 1.1771

PREGUNTAS: Incluye en la memoria de la práctica las respuestas a las siguientes preguntas:

- **¿Cuál es el rendimiento de cada uno de los modelos?**

El primer modelo obtiene un ERR de 2.1792 y el segundo de 1.1771

- **¿Cuál es mejor? ¿Por qué?**

Siguiendo los recientes resultados comentados queda claro que **el segundo** modelo (V2) es mejor.

- **¿Qué es el EER? ¿Cómo podemos interpretarlo?**

ERR son las siglas de Equal Error Rate, es decir, donde la tase de falsa aceptación (FAR) y de falso rechazo (FRR) se igualan o se interceptan.

Es la medida más eficaz para explicar el rendimiento de modelos, ya que simplemente diciendo la cantidad del mínimo ERR, valores más bajos de FAR y de FRR posibles conjuntamente. Es posible reducir uno de los dos, pero el otro siempre incrementara, por ello es tan buena medida.

- **¿Cuáles son las diferencias entre ambos modelos?**

La mayor diferencia la encontramos en elcnúmero de parámetros entrenados:

Model 1: 1437080 Vs. Model 2: 11103416

4. Entrenamiento de un modelo

Utilizando el script trainSpeakerNet.py, entrena dos extractores de embeddings (x-vectors) a partir del subconjunto dado en data/voxceleb2. Ten en cuenta que el entrenamiento puede llevar bastante tiempo dependiendo del hardware disponible. Puedes parar el entrenamiento fijando un número máximo de iteraciones (épocas).

COMANDOS:

Train:

```
!python ./trainSpeakerNet.py --model ResNetSE34L --log_input True --max_epoch 10 \
--encoder_type SAP --batch_size 200 --n_mels 64 --trainfunc amsoftmax --scale 30 --margin 0.3 --nClasses 4953 \
--save_path exps/results --eval_frames 400 --test_list lists/veri_test.txt --train_list lists/train_list.txt
```

Test:

```
!python ./trainSpeakerNet.py --eval --model ResNetSE34L --log_input True --max_epoch 10 \
--encoder_type SAP --batch_size 200 --n_mels 64 --trainfunc amsoftmax --scale 30 --margin 0.3 --nClasses 40 \
--save_path exps/results --eval_frames 400 --test_list lists/veri_test.txt --train_list lists/train_list.txt \
--initial_model exps/results/model/model000000010.model
```

PREGUNTAS: Responde a las siguientes preguntas:

- **¿Cuántas épocas has usado?**

5

- **¿Cuál es el rendimiento del modelo?**

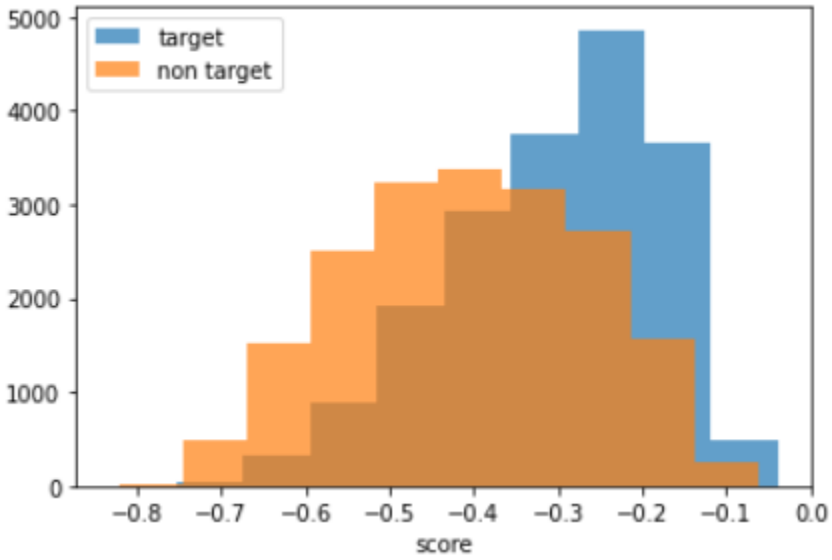
Bastante malo:

```
LR 0.001000, TEER/TAcc 0.00, TLOSS 16.702835, VEER 35.6999, MINEER 35.6999
EER = 35.6999
```

- **¿Cómo es este rendimiento en comparación con el analizado en el apartado anterior de los modelos preentrenados? ¿Por qué piensas que ocurre esto?**

Mucho peor. Esto seguramente se deba al número de épocas que ha sido entrenado.

- **Incluye los histogramas de tu sistema en la memoria de la práctica. ¿Qué umbral elegirías (valor aproximado) para clasificar entre target y non-target trials? ¿Por qué?**



4.2. Entrenamiento de un modelo

Ahora, selecciona un modelo de aquellos definidos en models/ así como una función de coste de las disponibles en la carpeta loss/. Los argumentos que necesitarás para entrenar dicho modelo dependerán de la configuración elegida.

PREGUNTAS: Contesta las siguientes preguntas en la memoria de la práctica:

- **¿Qué comando (modelo, parámetros, etc.) has utilizado?**

!python ./trainSpeakerNet.py --model VGGVox --log_input True --max_epoch 5 \

--encoder_type SAP --trainfunc amsoftmax --nClasses 4953 \

--save_path exps/optional1 --eval_frames 400 --test_list lists/veri_test.txt --train_list lists/train_list.txt --nOut 256

- **¿Cuál es el rendimiento del sistema?**

Train: LR 0.001000, TEER/TAcc 0.00, TLOSS 23.477459

Test: EER 44.5599

- **¿Cómo es el rendimiento de este sistema con respecto al resto de modelos de la práctica? ¿Por qué?**

Es peor, de entrada, es el modelo entrenado con menor número de épocas, por otro lado, los parámetros pueden ser óptimos siendo los predeterminados que otros pensados específicamente para optimizar el modelo respecto al problema.

Por último, entrena este mismo modelo, pero en una configuración con menos parámetros (modelo más pequeño) y evalúalo.

PREGUNTAS: Incluye las respuestas a estas preguntas en tu informe de la práctica:

- **¿Cómo es el rendimiento de este modelo pequeño con respecto al grande? ¿Por qué?**

Este es el rendimiento del modelo grande:

Train: LR 0.001000, TEER/TAcc 0.00, TLOSS 23.543533

Test: ERR 42.6988

El modelo pequeño es algo peor. Esto se debe a que al ampliar el tamaño del embeding conseguimos añadir más información.

- **Dibuja el histograma de scores para ambos modelos (pequeño y grande).**

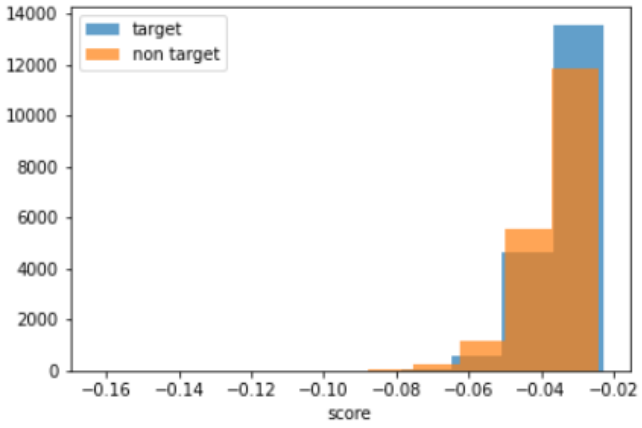


Figura 1 – Modelo Pequeño

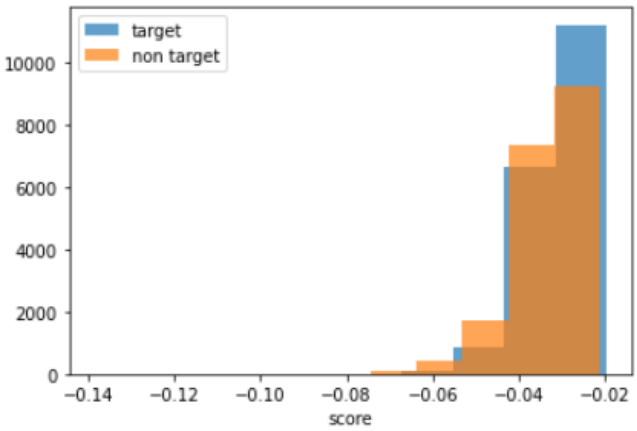


Figura 2 – Modelo Grande