REDES NEURONALES PARA BCIS

Diego Forniés, Iker Rabanal, Unai Salaberria
Facultad de Informática
Universidad del País Vasco
Donostia - San Sebastián

ABSTRACT

Las interfaces cerebro-computadora (BCI, por sus siglas en inglés) se utilizan para transformar señales eléctricas producidas por el cerebro en órdenes. Las señales eléctricas deben ser decodificadas para obtener la orden. Estas interfaces son muy útiles para dar asistencia a personas en el proceso de rehabilitación de funciones perdidas.

Nuestro objetivo en este proyecto será clasificar esas señales en las órdenes concretas. Para ello, vamos a utilizar diferentes arquitecturas basadas en redes neuronales, con el objetivo de clasificar las señales en órdenes concretas.

1. Introducción

El problema sobre el que vamos a trabajar tiene como objetivo clasificar las señales en distintas órdenes motrices. Es decir, la señal eléctrica detectada en el cerebro del sujeto debe ser transformada a un movimiento físico. Normalmente, el cuerpo humano ignora este proceso, pero en realidad funciona de la siguiente forma: el cerebro genera una orden en forma de señal eléctrica (consciente o inconsciente), y esta es transmitida hasta el músculo concreto que debe realizarla.

Nuestra tarea consistirá en realizar la función de transformación de la señal, tendremos que detectar qué orden o acción se quiere llevar a cabo partiendo de una señal eléctrica.

Con todo esto en mente, iremos probando distintas arquitecturas y algoritmos, para encontrar la más apropiada en esta tarea de clasificación.

Para poder seguir la lógica del informe y ver los resultados obtenidos, pudiendo también reproducirlos, dejamos aquí el link al cuaderno de Colab en el que hemos desarrollado el proyecto:

`https://colab.research.google.com/drive/1I7MegpRRwthGUg6wH6j7PPjrfjoujL-o?usp=sharing`

2. Descripción del problema

Una interfaz cerebro-computador (BCI) es un sistema que establece la comunicación directa entre el cerebro y un ordenador. La actividad eléctrica, producida cuando las neuronas se disparan para comunicarse entre sí, se procesa y decodifica en comandos que se envían a un dispositivo para llevar a cabo una acción deseada.

Estas acciones cubren diversas aplicaciones, desde rehabilitación de funciones perdidas hasta el aumento de nuestras capacidades.

En nuestro caso, el objetivo es buscar el modelo que mejor consiga distinguir qué es lo que está pensando cada sujeto. Las posibles opciones son:

- Imaginar mover la mano derecha.
- Imaginar mover la mano izquierda.
- Imaginar una palabra que empiece por una misma letra.

3. Descripción de los datos

3.1. Datos

Para recolectar los datos se realizaron cuatro sesiones con tres sujetos diferentes, y por cada sujeto se recopilaron dos tipos de datos: los encefalogramas y las características preprocesadas. Los encefalogramas contienen la señal eléctrica de cada sujeto (EEG). A partir de ellas, aplicamos un filtro superficie de Laplace y obtenemos la densidad espectral de frecuencia, de donde obtenemos las características preprocesadas.

Como ya sabemos, vamos a tener que clasificar esta información en base a la orden que ha imaginado el sujeto. Recordamos las tres etiquetas diferentes:

- La primera, cuando el sujeto imagina mover la mano derecha.
- La segunda, cuando el sujeto imagina mover la mano izquierda.
- La tercera, cuando el sujeto imagina una palabra que empiece por una misma letra.

Para recolectar estos datos cada uno de los tres sujetos estuvo sentado, con los brazos apoyados en las piernas durante cuatro minutos seguidos. Tras un descanso de diez minutos, los sujetos vuelven a hacer lo mismo, hasta llegar a cuatro sesiones.

3.2. Formato de los datos

Una vez explicado cómo han sido recopilados los datos, conviene especificar el formato de los datos.

3.2.1. Señales EEG (Electroencefalograma)

Los datos EEG obtenidos en cada prueba han sido muestreados a una frecuencia de 512 hercios. Como a cada parte de nuestro cerebro se le asigna una función diferente, se han puesto 32 electrodos en diferentes partes del cuero cabelludo, para así detectar la actividad cerebral de cada zona. Por ello, este primer dataset va a tener 32 dimensiones por cada muestra, más una 33º que asigna la clase a la que pertenece la muestra.

Específicamente, las ubicaciones de los electrodos son Fp1, AF3, F7, F3, FC1, FC5, T7, C3, CP1, CP5, P7, P3, Pz, PO3, O1, Oz, O2, PO4, P4, P8, CP6, CP2, C4, T8, FC6, FC2, F4, F8, AF4, Fp2, Fz, Cz.

3.2.2. Características Preprocesadas

Estas características derivan de los datos anteriores. Se aplica un filtro de superficie de Laplace a los datos EEG, que consigue enfatizar la diferencia de actividad eléctrica en electrodos cercanos, y así favorecer a la detección de patrones.

Cada 62.5 ms, lo que equivale a 16 veces por segundo, se estima la Densidad Espectral de Potencia (PSD) en el rango de frecuencia de 8 a 30 Hz utilizando los datos de EEG del último segundo. Esta estimación se ha hecho con una resolución de frecuencia de 2 Hz y se ha aplicado a ocho canales ubicados en diferentes regiones del cuero cabelludo, específicamente en los canales C3, Cz, C4, CP1, CP2, P3, Pz y P4.

Como resultado, una muestra de estas características preprocesadas se representa como un vector de 96 dimensiones (12 frecuencias x 8 canales), más la 97º perteneciente a su clase.

3.3. Dataset

El conjunto de datos se nos ha proporcionado en la siguiente página:

`https://www.bbci.de/competition/iii/#references`

Nosotros hemos elegido el dataset V.

El dataset ha sido obtenido de tres sujetos diferentes. Como cada uno tenía 4 sesiones diferentes, con tres de ellas entrenaremos y con la última evaluaremos el modelo.

Antes de empezar a entrenar modelos, hemos separado el conjunto de entrenamiento en tres: un conjunto para entrenar el modelo, un conjunto de validación, con el que podremos tomar decisiones como cambiar el valor de los hiperparámetros, y un tercer conjunto de test, ya que el conjunto que se nos proporciona no es del todo correcto, como explicaremos en el apartado 5.

	Entrenamiento	**Test**
Datos EEG	1096192	365056
Car. prepr.	31216	10464

Cuadro 1: División del dataset

4. Diseño e implementación

En el camino a encontrar el modelo que mejor clasifique las señales del cerebro, hemos planteado muchas posibles arquitecturas y modelos. Además, hemos leído [1] antes de implementar los modelos por si encontrábamos algo que nos fuera a ser útil.

4.1. Arquitectura del modelo

La primera duda que nos surgió, al tener dos tipos de inputs diferentes, fue cómo podíamos sacarle el máximo partido a los datos: utilizando ambos o utilizando sólo uno de los dos.

4.1.1. Concatenación de input

Una opción era unir la señal EEG y las características preprocesadas (que llamaremos CP) en un único tensor, e introducir el vector resultante en el modelo. Esta idea no nos convenció, ya que para cada muestra de las CP necesitamos 256 muestras de la señal EEG.

Por ello, la diferencia de elementos de cada input es enorme, y podría ser que el modelo no sacara el máximo partido a las CP.

4.1.2. Un modelo para cada input y unir

Debido al problema anterior, pensamos alguna manera de procesar cada input, y obtener un vector de la misma longitud en ambas entradas. Para ello, creamos una arquitectura con dos modelos diferentes, para que cada uno procese un input, y posteriormente concatenar las salidas e introducirlas en un nuevo modelo. Así, cada uno obtendría un vector de características de las mismas dimensiones, y al concatenarlos e introducirlos en otro modelo, éste podría procesar y aprender mucho mejor la información dada.

Para ello, planteamos utilizar una MLP o RRN para las señales EEG, y una MLP para las CP. Finalmente, tras su concatenación, pensamos que lo mejor podría ser una MLP.

Al intentar implementar esta arquitectura, nos surgieron varios problemas, de los que hablaremos posteriormente.

4.1.3. Utilizar sólo EEG

Otra opción sencilla sería utilizar únicamente la señal EEG de entrada, procesarla en un modelo, y obtener su clasificación. Para ello, planteamos utilizar una MLP, una RRN o un Transformer.

4.1.4. Utilizar sólo CP

La última opción planteada fue parecida a la anterior, pero utilizando únicamente las características preprocesadas de entrada. Para ello, planteamos utilizar una MLP o un Transformer.

4.2. Implementación y resultados

Para crear un modelo basado en redes neuronales, hemos decidido seguir los consejos que ofrece Andrej Karpathy en su entrada de blog: A recipe for Training Neural Networks [2].

Como menciona, una vez comprendemos bien la estructura de los datos, y sabemos que se puede entrenar y evaluar un modelo sobre ellos, la aproximación que mejor va a funcionar es sobreajustar un modelo grande e ir regularizándolo poco a poco.

4.2.1. Modelo EEG

Tras hacer una primera prueba con diferentes arquitecturas, vimos que procesar la señal EEG mediante una MLP podía ser una solución poco costosa y efectiva.

Nuestro primer modelo va a tener la siguiente estructura:

$[LINEAR \rightarrow BNORM \rightarrow RELU] * 2 \rightarrow FC$

Además, utilizamos la inicialización de pesos Kaiming para cada una de las capas lineales.

Aplicando varios de los consejos, como por ejemplo utilizar Adam como optimizador, o no meter mucha complejidad en un paso directamente, hemos conseguido un resultado de entrenamiento bastante bueno. Pero, como el resultado sobre el conjunto de validación también es bueno, no podemos decir que haya sobreajuste. En la siguiente tabla podemos ver los resultados tanto de la fase de entrenamiento (tras 10 épocas), como los de la fase de validación:

	Entrenamiento	**Validación**
Pérdida	0.3	0.54
Acierto	87.75 %	79.35 %

Cuadro 2: Primeros resultados de la Red Neuronal

Aunque la diferencia entre los resultados sea baja, y no haya un gran sobreajuste, vamos a probar a utilizar varias técnicas de regularización, a ver si conseguimos mejorar los resultados.

Pero antes de pasar a regularizar, hemos probado a añadir un planificador del learning rate, a ver si se consigue alguna mejora antes del siguiente paso.

Los resultados han sido similares, incluso algo peores, utilizando el planificador. El modelo con planificador de learning rate ha obtenido un acierto del 87.68 % y una pérdida de 0.31 en el conjunto de entrenamiento, y un 75.39 % y 0.78 en el conjunto de validación.

Por tanto, vamos ahora a regularizar el modelo sin planificador.

Regularizar el modelo

Con el objetivo de regularizar el modelo actual, vamos a aplicar tres cambios:

1. Añadir decaimiento de pesos en el optimizador Adam.
2. Añadir dilución en cada capa linear de nuestro modelo.
3. Reducir el tamaño de batch.

Los resultados de la regularización han sido muy malos: 37.88 % de acierto y 1.07 de pérdida en el conjunto de entrenamiento, y 40.17 % de acierto y 1.07 de pérdida en el de validación.

Queda claro que nuestra mejor opción es utilizar el primer modelo, y quizás añadir alguna capa más a la arquitectura y entrenarlo más tiempo, antes de pasar a la fase de test.

Red más profunda y entrenada

Una vez tenemos claro que nuestra mejor opción va a ser la primera arquitectura que hemos definido, vamos a intentar mejorar ese primer resultado de acierto del 79.35 % que hemos obtenido del conjunto de validación.

En un principio, sólo hemos añadido una capa más a la arquitectura, así que quedará tal que así:

$[LINEAR \rightarrow BNORM \rightarrow RELU] * 3 \rightarrow FC.$

Veremos si esta arquitectura obtiene buenos resultados o, por el contrario, tenemos que añadir más capas.

El proceso de entrenamiento ha sido de 50 épocas, y después se ha ejecutado el modelo sobre el conjunto de validación. Una vez hemos visto que los resultados sobre el conjunto de validación han sido muy buenos, hemos dado por válido el modelo, y lo probamos sobre el conjunto de test que separamos al principio.

En la siguiente tabla tenemos los resultados de este nuevo modelo:

	Entrenamiento	**Validación**	**Test**
Pérdida	0.09	0.2	0.2
Acierto	96.48 %	93.09 %	93.01 %

Cuadro 3: Resultados de la Red Neuronal más profunda y entrenada

4.2.2. Modelo con las características preprocesadas

A pesar de que los resultados con el modelo sobre los datos EEG son muy buenos, hemos decidido probar también a crear uno utilizando esta vez las características preprocesadas. Ya hemos comentado antes que estas características se derivan de las señales EEG, por lo que deberíamos conseguir resultados parecidos.

El formato de los datos preprocesados es muy similar a los EEG, así que hemos probado a utilizar el mismo primer modelo, con tres capas y utilizamos la normalización por lotes, la función de activación ReLU y la inicialización de pesos Kaiming.

Para recordarlo, esta es la estructura de este modelo inicial:

$[LINEAR \rightarrow BNORM \rightarrow RELU] * 2 \rightarrow FC$

Nos ha sorprendido que, con estos datos, los resultados han sido prácticamente iguales. En la siguiente tabla los vemos, tanto en la fase de entrenamiento como en la de validación:

	Entrenamiento	**Validación**
Pérdida	0.34	0.44
Acierto	87.08 %	81.63 %

Cuadro 4: Resultados del primer NN sobre las características preprocesadas

Red más profunda y entrenada

Los resultados son muy buenos, así que vamos a probar nuevamente a utilizar un modelo más profundo y entrenado. Esta vez, vamos a omitir la parte de regularización del modelo, visto que antes no ha tenido éxito.

El modelo que vamos a utilizar es el mismo que desarrollamos para los datos EEG. Para recordar, esta es la estructura que tiene la arquitectura:

$[LINEAR \rightarrow BNORM \rightarrow RELU] * 3 \rightarrow FC.$

Además, como antes, vamos a dejarlo entrenar durante 50 épocas.

Tras el proceso de entrenamiento, hemos seguido las mismas pautas que con los datos EEG. La prueba sobre el conjunto de validación ha dado muy buenos resultados, así que hemos pasado a la fase de test.

En la siguiente tabla tenemos todos los resultados del modelo sobre los datos de características preprocesadas:

	Entrenamiento	**Validación**	**Test**
Pérdida	0.11	0.04	0.29
Acierto	96.05 %	99.02 %	90.26 %

Cuadro 5: Resultados de la Red Neuronal profunda sobre las características preprocesadas

4.2.3. Modelo combinado EEG y características preprocesadas

Por último, vamos a probar a combinar los dos inputs. Para ello, hemos implementado una arquitectura basada en los modelos anteriores.

Visto que las MLPs implementadas trabajan muy bien, hemos utilizado una para procesar cada entrada, y que su salida sea un vector de las mismas longitudes para ambos. Así, conseguimos un embedding para cada input con las mismas medidas, y posteriormente los concatenamos para introducirlo en un último MLP, similar a los anteriores.

En este caso, hemos utilizado un tamaño de embedding de 32 para cada entrada, y un hidden size de 64 en las redes neuronales.

Procesamiento de los datos

Al intentar procesar los datos, tuvimos varios problemas.

Para entrenar el modelo, necesitamos introducir datos del mismo instante en ambas entradas. Entendiendo el dataset, vemos que necesitamos 512 muestras EEG para una muestra preprocesada, ya que la última hace un espectro del último segundo de la señal EEG, muestreadas a 512 hercios. Además, la frecuencia de muestreo de las características preprocesadas es de 16 hercios, por lo que hay que coger 512 muestras cada 32 para ir alineándolas con estas.

Al añadir esta función, nos dimos cuenta que el dataset no es del todo exacto, debido a que siempre sobraban algunas muestras EEG. Para solucionarlo, decidimos ignorar las muestras sobrantes.

Resultados

Los resultados de este modelo son muy buenos, ya que consigue un accuracy muy alto en todas las particiones. Lo hemos entrenado con 50 épocas, para su posterior validación. Al ver que el modelo no estaba sobreajustado, evaluamos su acierto final en el test. Estos han sido los resultados finales:

	Entrenamiento	**Validación**	**Test**
Pérdida	0.16	0.17	0.23
Acierto	94.4 %	94 %	92.15 %

Cuadro 6: Resultados del modelo implementado sobre las señales EEG y las características preprocesadas

4.2.4. Otros modelos

Durante la implementación de estos modelos, pensamos otro tipo de soluciones, que finalmente no fueron implementadas.

LSTM

Este tipo de red neuronal recurrente fue una de nuestras primeras ideas al ver que, en el modelo combinado, debíamos agrupar varias muestras de la señal EEG.

Sin embargo, tras implementar una primera versión básica, vimos que los resultados no eran los esperados y el entrenamiento era bastante lento, por lo que optamos por descartar esta opción y centrarnos en otras.

Transformer

La red neuronal Transformer también fue planteada al inicio del proyecto, ya que el uso de la atención en este problema podría dar resultados excelentes.

Aún así, tras ver los buenos resultados que nos estaban dando opciones más sencillas y menos costosas, decidimos que no era la mejor opción. Tal vez hubiera dado resultados un poco mejores, pero el costo computacional del modelo no saldría rentable con la leve mejora.

5. Conjunto de test

Para el problema, se nos proporcionaba un conjunto de test para cada sujeto. De las cuatro sesiones que llevaron a cabo los sujetos, una se separó para ser utilizada como conjunto de test.

Las etiquetas de estos conjuntos de test no se nos daban de forma directa, sino que en una página a parte estaban todas. Por lo que, lo primero que tuvimos que hacer fue procesar estas etiquetas para guardarlas en el formato correcto.

Una vez hecho esto, el enunciado nos dice que el conjunto de test debe ser procesado de distintas formas dependiendo de a qué tipo de datos pertenezca:

- Datos EEG: Si el conjunto de test proviene de los datos EEG, entonces hay que procesar los vectores con los datos 16 veces por segundo, y utilizar los datos del último segundo. Después, hay que calcular la media de ocho muestras consecutivas.
- Características preprocesadas: Aquí simplemente tenemos que calcular la media de ocho muestras consecutivas.

Primero lo probamos con los datos EEG, pero tras procesar los datos tal y como dice el enunciado, nos dimos cuenta que teníamos menos etiquetas de las necesarias: 1390 etiquetas entre los tres sujetos, frente a 1426 vectores de datos EEG. Aunque la falta de etiquetas no sea demasiado grande, supone un problema muy grave, ya que no sabemos dónde faltan.

Es decir, puede que esas 36 etiquetas falten todas al inicio, al final o estén intercaladas por todo el conjunto de test.

Por tanto, lo más probable es que el modelo clasifique los datos según lo que ha aprendido con las anteriores sesiones sobre el mismo sujeto, y en cambio, la etiqueta se esté refiriendo a una muestra distinta.

En definitiva, creemos que el conjunto de test tiene errores visibles, por lo que la clasificación con él no va a ser fiable. Aún así, hemos probado a ejecutar el modelo, y los resultados han sido muy parecidos a lo que conseguiríamos haciendo una clasificación aleatoria, que sería un acierto cercano al 33 %.

Es por eso que decidimos guardar un conjunto de test desde el dataset inicial, para poder comprobar nuestro modelo sobre datos fiables.

6. Conclusiones

Tras haber estudiado el problema de clasificación de las BCIs y haber implementado diferentes modelos, hemos obtenido las siguientes conclusiones.

Por un lado, creemos que la utilización de la señal EEG puede ser suficiente para abordar problemas de este tipo. Los resultados del modelo utilizando únicamente esta entrada han sido excelentes, y por ello no es del todo necesario añadir otros datos preprocesados añadidos a partir de éstos.

Por otro lado, el tipo de modelo a implementar puede variar dependiendo de la complejidad del problema. En este caso, al existir únicamente 3 clases, un modelo MLP profundo ha sido suficiente para conseguir buenos resultados. Sin embargo, en otros problemas donde la complejidad sea mucho mayor y se necesite predecir entre muchas salidas diferentes, quizá sea necesario implementar un modelo más costoso y complejo.

Si tuvieramos que implementar un modelo real para este problema, nuestra elección sería sin duda el modelo basado únicamente en la señal del electroencefalograma. Esto se debe a diferentes razones.

Para empezar, su coste computacional es más bajo que el modelo combinado, y consigue resultados muy parecidos. Además, los datos no tienen que ser procesados, y desaparece el problema de alineación con las características preprocesadas. Por último, los datos de entrada son obtenidos directamente de los electrodos, y no necesitan ser tratados, lo cual también requiere de costo operacional.

Por todo esto, creemos que es el modelo que mejor se ajusta a las necesidades del problema, y si comparamos en base a la calidad y el coste, es el más destacado.

Referencias

[1] Fabien Lotte, Marco Congedo, Anatole Lécuyer, Fabrice Lamarche, and Bruno Arnaldi. A review of classification algorithms for eeg-based brain–computer interfaces. *Journal of neural engineering*, 4(2):R1, 2007.

[2] Andrej Karpathy. A recipe for training neural networks.

[3] Ferran Galán, Francesc Oliva, and Joan Guardia. Bci competition iii. data set v: Algorithm description. *Brain computer interfaces competition III*, 2005.