AAI - Tema 5: Introducción a las redes neuronales - Teoría

Descenso del gradiente

- Descenso del gradiente
  - Implementación en Scikit-learn
    - `SGDClassifier`
      - Complejidad computacional $O(m \times n)$

El descenso del gradiente se explica en el capítulo 4 de HOML. Es conveniente leer este capítulo previamente a redes neuronales ya que las referencias son numerosas.

- Bibliografía
  - Capítulo 4. HOML. 2.ª ed.

Redes neuronales

- Ficha
  - Requiere escalado: sí
  - Supervisión: sí
  - Vol. instancias: alta
- Redes neuronales
  - Tipos de redes neuronales
    - Perceptrón
    - Perceptrón multi-capa (MLP)
- Algoritmo de retroprogación
- Funciones de activación
  - Heaviside
  - Signo
  - Sigmoide / Logística
  - Tangente hiperbólica (tanh)
  - ReLU
  - Softplus
  - Softmax

Perceptrón

La **regla de Hebb** es la inspiración histórica del perceptrón y se resume como que en las neuronas hay un aumento del peso con co-activación, es decir, que hay conexión entre dos neuronas tiende a fortalecerse si se activan a la vez.

Se preguntó en EX.20250204.T.9.

Perceptrón multicapa (MLP)

Ventajas del MLP:

- Capacidad para aprender **modelos no lineales**.
- Capacidad para aprender modelos en tiempo real (**aprendizaje *on-line***) empleando `partial_fit`.

Desventajas:

- Las capas ocultas tienen una función no convexa donde **existe más de un mínimo local**. Por ello, distintas inicializacionalices de peso pueden llevar a distinta exactitud de validación.
- Es **sensible al escalado** de características.

La **complejidad computacional** del MLP viene dada por:

$$O(i \cdot n(m \cdot h + (k-1) \cdot h \cdot h + h \cdot o))$$

donde:

- n son las instancias de entrenamiento
- m las características
- k las capas ocultas, cada una de ellas conteniendo
- h neuronas (simplificando) y o neuronas de salida

Respecto a la **transformación de datos**, es necesario aplicar un **escalado** común a todos los conjuntos de datos. Por ejemplo, escalar cada atributo del vector de entrada X a [0, 1] ó [-1, +1], estandarilizarlo para que haya media 0 y varianza 1.

A esto puede contribuir la clase `StandardScaler` de Scikit-learn.

Bibliografía:

- Para ventajas y desventajas del MLP: Apartado 1.17.1 de Scikit-learn.
- Para la complejidad computacional del MLP: Apartado 1.17.6 "Complexity" en Scikit-learn.
- Para transformación de datos de MLP: el primer punto del Apartado 1.17.7 "Tips on practical use" en Scikit-learn.

Dimensiones matriciales en MLP

Se asume que la convención que se usa en la asignatura es la que se corresponde con la práctica, que coincide con la representada en la página 305 del libro HOML 3.ª ed.:

$$\hat{Y} = \phi(X\,W + b)$$

donde:

- La matriz W tiene una fila por entrada y una columna por neurona.

Es la misma convención que usan frameworks como PyTorch y TensorFlow.

Algunos libros teóricos o chatbots en ocasiones ofrecen la fórmula $\hat{Y} = \phi(W\,W + b)$, pero en la asignatura no se sigue esta convención.

Ejemplo de MLP:

- Instancias en lote: $M = 2$
- Entradas: $D = 10$
- Neuronas en capa oculta: $H = 50$
- Neuronas en capa de salida: $K = 3$

Cálculo de salida de una capa oculta:

$$h = f(XW_h + b_h)$$

donde:

- X: matriz de entrada (el lote)
- W_h: matriz de pesos de la capa oculta
- b_h: vector de sesgos (*bias*) de la capa oculta
- h: salidas de la neuronas ocultas

Matriz/vector	Notación	Ejemplo
Entrada	$X \in \mathbb{R}^{M \times D}$	2×10
Pesos capa oculta	$W_h \in \mathbb{R}^{D \times H}$	10×50
Sesgos capa oculta	$b_h \in \mathbb{R}^{1 \times H}$	1×50
Salida capa oculta	$h \in \mathbb{R}^{M \times H}$	2×50

Cálculo de salida final:

$$Y = g(h\,W_o + b_o)$$

donde:

- h: salidas de la neuronas ocultas
- W_o: matriz de pesos de la salida final
- b_o: vector de sesgos (*bias*) de la salida final
- Y: salidas finales

Matriz/vector	Notación	Ejemplo
Salida capa oculta	$h \in \mathbb{R}^{M \times H}$	2×50
Peso salida final	$W_o \in \mathbb{R}^{H \times K}$	50×3
Sesgo salida final	$b_o \in \mathbb{R}^{1 \times K}$	1×3
Salida final	$Y \in \mathbb{R}^{M \times K}$	2×3

Es posible que ver esta fórmula con la matriz de pesos representada como como $W^\top$ en lugar de W. Se trata solo de una diferencia de convención; en cualquier caso las filas y columnas de la matriz que aparece en la fórmula son las mismos.

La multiplicación de matrices restringe que:

- Las columnas de la primera de matriz multiplicada deben ser iguales a las filas de la segunda matriz multiplicada.
- La matriz resultante tiene un número de filas igual a las filas de la 1.ª matriz y un número de las columnas igual a las columnas de la 2.ª matriz.

En caso de que se pida indicar las dimensiones de una de las matrices dado un escenario, entre las fórmulas y las restricciones de las multiplicaciones de matrices (necesario que) y las dimensiones que tienen que tener la matriz de resultado () al final es inferir la única opción posible

Preguntado en AAI.EX.20200201.T.9, EX.20200901.T.9, EX.20230207.T.7 AAI.EX.20240206.T.5 y EX.20240902.T.10.

Entrenamiento

La **retropropagación** (backpropagation) es el algoritmo fundamental de ANN. Su objetivo es ajustar los pesos de las conexiones para minimizar una **función de pérdida** que mide el error entre la salida predicha y la real.

Fases (tradicionalmente son 2, ya que el tercero lo hace un optimizador):

1. "paso hacia adelante" (*forward pass*) donde los datos atraviesan la red para generar un resultado
2. **paso hacia atrás** (*backward pass*) el cálculo del gradiente del error respecto a cada peso mediante la **regla de la cadena**. Este gradiente se propaga desde la capa de salida hacia atrás hasta la capa de entrada, determinando cuánto "crédito" o responsabilidad tiene cada peso en el error total.
3. Actualización de peso (*weight update*) Algoritmo de optimización, como el **descenso de Gradiente**, para actualizar los pesos en la dirección opuesta al gradiente, reduciendo así el error de forma iterativa hasta que la red converge en una solución óptima.

Paso hacia atrás

Es el algoritmo para calcular el gradiente. Su herramienta es la derivada.

Para calcular el **descenso del gradiente** es conveniente que la función sea derivable en todos sus puntos.

El **problema del gradiente desvanecido** o *vanishing gradient* en el algoritmo de propagación es que los gradientes se vuelven cada vez más pequeños a medida que retroceden por las capas hasta ser casi cero.

Se debe evitar derviar cero por el problema del gradiente desvanecido

Algoritmo de optimización

El algoritmo de optimización puede ser:

- Descenso del gradiente
- Adam

En **descenso del gradiente**, los pesos se actualizan restando al valor actual una fracción del gradiente del error (determinada por la tasa de aprendizaje), lo que empuja los pesos en la dirección que minimiza la pérdida total del modelo.

$$w_j \leftarrow w_j - \eta \frac{\partial J}{\partial w_j}$$

donde:

- w El peso que queremos ajustar.
- η (eta): La tasa de aprendizaje (learning rate), que controla qué tan grande es el paso que damos hacia el mínimo.
- $\frac{\partial J}{\partial w_j}$: El gradiente (derivada parcial), que indica la dirección y magnitud del error respecto a ese peso específico.

Funciones de activación

Las capas ocultas requieren que la activación sea no lineal, ya que de lo contrario el resultado es equivalente al de tener una sola capa.

- Heaviside
- Signo
- Sigmoide / Logística
- Tangente hiperbólica (tanh)
- ReLU
- Softplus
- Softmax

Función activación	Rango	Continuidad	Uso
Sin activación	N/A	Discreto	Salida con valores sin acotar
Heavside	(0,1)	Discreto	-
Signo	(-1,1)	Discreto	-
Sigmoide/logística	(0,1)	Discreto	Clasificación +multiclase
tanh	(-1,1)	Continuo	Capas ocultas (no linear)
ReLu	(0,z)	Continuo	Capas ocultas (no linear)
Softplus	(0,z)	Continuo	Capas ocultas (no linear)
Softmax	Varias salidas	Continuo	Clasificación multiclase

Heaviside

$$H(z) = \begin{cases} 0 & z < 0 \\ 1 & z \geq 0 \end{cases}$$

Signo

Llamada signo o *sign*.

$$\operatorname{sgn}(z) = \begin{cases} -1 & z < 0 \\ 0 & z = 0 \\ +1 & z > 0 \end{cases}$$

Sigmoide / Logística

La función sigmoide o logística es una función de activación con forma de S:

$$\sigma(z) = 1/(1 + \exp(-z)).$$

Tiene un valor de salida entre 0 y 1:

$$\sigma(z) \in (0, 1).$$

La función tiene una derivada distinta de cero definida en todos los puntos, permitiendo al descenso del gradiente (GD) progresar en cada paso.

Rumelhart fue el primero en aplicarla.

Usos:

- **Clasificación binaria**, pues tiene un valor de salida entre 0 y 1.
- **Clasificación binaria multi-etiqueta**, pues permite que múltiples neuronas tengan un valor próximo a 1, a diferencia de softmax en en el que todos deben sumar uno.

A pesar de ser no lineal, en la práctica tiene valores próximos a 0 y 1, lo que produce saturación y el problema del gradiente desvanecido por lo que no es apto para su uso en capas ocultas.

Para entradas pequeñas suele producir valores cercanos a 0.5.

Tangente hiperbólica (tanh)

La tangente hiperbólica es una función de activación con forma de S.

$$\tanh(z) = 2\sigma(2z) - 1$$

Da un valor entre -1 y 1.

Al estar centrada en 0 produce valores pequeños positivos o negativos alrededor de ese punto.

Es no lineal, por lo que se usa en capas ocultas especialmente.

ReLU

La función de unidad linear rectificada o *rectified linear unit* (ReLU) no es realmente continua. Da un valor que es ó 0 ó z:

$$\text{ReLU}(z) = \max(0, z).$$

En la práctica, es computacionalmente eficiente y da buenos resultados.

Es no lineal, por lo que se usa en capas ocultas especialmente.

Softplus

$$\text{softplus}(z) = \log(1 + \exp(z))$$

Softplus es una variante suave de ReLU. Se acerca a 0 cuando es negativa, y a z cuando es positiva.

Da un valor continuo entre 0 y zs.

Softmax

Softmax usa un vector de neuronas completo en lugar de una sola.

Su salida es un vector de probabilidad

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Usos:

- **Capa de salida de clasificación multiclase**.