Ayudantía
SCT - Semana - 02
Estándar de punto flotante y pérdida de importancia.

Función de activación:

En el núcleo de las Redes Neuronales Artificiales se encuentran las funciones de activación. Estas tiene como objetivo introducir una dependencia no-lineal entre la/s variables de entrada y la/s variables de salida. Algunos ejemplos de funciones de activación son las siguientes:

Función escalón:	$H(x) = \begin{cases} 0, & \text{para } x < 0, \\ 1, & \text{para } 0 \leq x. \end{cases}$
Función logística:	$\sigma(x) = \dfrac{1}{1+\exp(-x)}$
Tangente hiperbólica:	$\tanh(x) = \dfrac{\exp(x)-\exp(-x)}{\exp(x)+\exp(-x)}$
Softplus:	$\log\left(1+\exp(x)\right)$
Sigmoid linear unit:	$\dfrac{x}{1+\exp(-x)}$

Tabla 1: Ejemplos de funciones de activación. Notar que $\log(x)$ corresponde al logaritmo natural.

En particular, se propone trabajar con una **variante** de la "Tangente Hiperbólica". Pero antes, realizaremos algunas modificaciones algebraicas, es decir,

$$f(x) = \frac{2^x - 2^{-x}}{2^x + 2^{-x}} = \frac{2^x}{2^x} \cdot \frac{1-2^{-2\,x}}{1+2^{-2\,x}} = \frac{1-2^{-2\,x}}{1+2^{-2\,x}}.$$

Más aún, se puede modificar un poco más la expresión anterior y obtenemos lo siguiente,

$$f_k(x) = \frac{1-2^{-2^k\,x}}{1+2^{-2^k\,x}}, \tag{1}$$

donde $f(x) = f_1(x)$, ver figura 1 para referencia. Adicionalmente se puede obtener la derivada de $f_k(x)$ con respecto a x, la cual es,

$$f_k'(x) = \frac{2^{2^k x+k+1}}{\left(2^{2^k x}+1\right)^2}\log(2),$$

es decir, es **no negativa**.

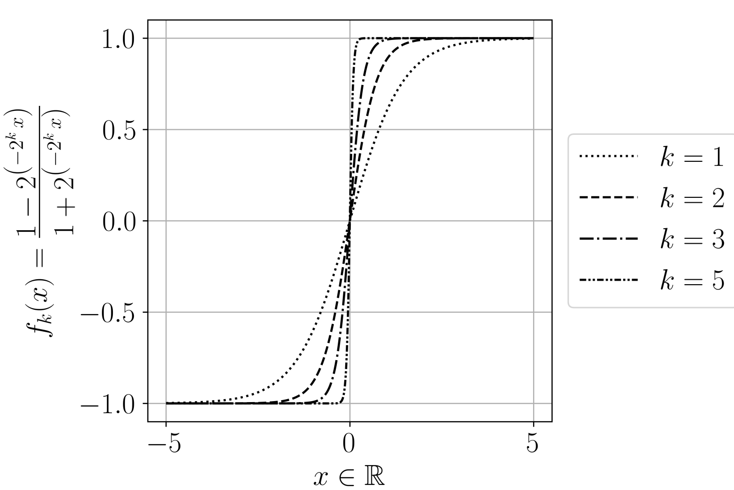

Figura 1: Gráfica de $f_k(x)$ para distintos valores de k en el dominio $[-5, 5]$.

Preguntas

(a) **[15 puntos]** Para la función $f_k(x)$, definida en la ecuación (1), considere el siguiente cambio de variable: $2^k\,x = m$, es decir reemplazamos el término $2^k\,x$ por la variable m. Particularmente estamos interesados en el conjunto de valores donde $m \in \mathbb{N}$. Es decir se obtiene la siguiente expresión:

$$\frac{1-2^{-2^k\,x}}{1+2^{-2^k\,x}} = \frac{1-2^{-m}}{1+2^{-m}} = \widehat{f}(m).$$

Pregunta: Determine, y justifique claramente, el **menor** valor de $m \in \mathbb{N}$ que asegure que $\widehat{f}(m)$ es exactamente igual a 1.0 si utilizamos *double precision* para la evaluación.

En caso de que fuera útil, se recuerda que el número 1.0 en el estándar de punto flotante para *double precision* se representa como:

$$1 = +1\,\textbf{.}\underbrace{0000000000000000000000000000000000000000000000000000}_{\text{mantisa con 52 0's}}\cdot 2^0,$$

o en su representación de máquina,

```
0 01111111111 0000000000000000000000000000000000000000000000000000
```

Se sugiere:

- Argumentar claramente utilizando el estándar de punto flotante IEEE 754.
- Justificar técnicamente cada una de las aproximaciones que realice.
- Fundamentar todos los puntos claves para recibir el puntaje completo.

Respuesta:

Análisis de los antecedentes:

- $m \in \mathbb{N}$, por lo cual estamos buscando el menor número entero m que genere $\widehat{f}(m) = 1.0$. Notar que la igualdad se logra en este caso cuando existe pérdida de importancia tanto en el numerador como en el denominador.
- Para que $\widehat{f}(m) = 1.0$ se logre, se necesita que tanto el numerador como el denominador de la expresión sean exactamente iguales en *double precision*.

Argumentación técnica:

- Sabemos que en *double precision* $\varepsilon_{\texttt{mach}} = 2^{-52}$, esto implica que el denominador es exactamente 1.0 para $m > 52$.
- Para que se logre la igualdad de los términos en este caso, se debe asegurar que tanto el numerador como el denominador sean iguales a 1.0.
- **[5 puntos]** El denominador es 1.0 cuando $m = 53$ dado que al sumar 2^{-53} a 1, la regla de redondeo indica que se debe truncar dado que el bit 52 es 0.
- **[5 puntos]** Para el caso del numerador cuando se usa $m = 53$ es aún representable, en particular se obtiene la siguiente representación en *double precision* para $\texttt{fl}(1-2^{-53})$,

  ```
  0 01111111110 1111111111111111111111111111111111111111111111111111
  ```

  es decir, es justo el antecesor representable del número 1.0. Entonces, si $m = 53$ la evaluación de $\widehat{f}(m)$ es,

  $$\begin{aligned}\frac{\texttt{fl}(1-2^{-53})}{\texttt{fl}(1+2^{-53})} &= \frac{\texttt{fl}(1-2^{-53})}{1.0}\\ &= 1-2^{-53}.\end{aligned}$$

  Lo cual es representable y distinto a 1.0, según lo solicitado.
- **[5 puntos]** La siguiente alternativa es $m = 54$. En este caso el denominador sigue siendo 1.0 dado que $\texttt{fl}(1+2^{-54}) = 1.0$, por lo tanto el análisis se centra en la evaluación de $\texttt{fl}(1-2^{-54})$. Según lo indicado anteriormente, $\texttt{fl}(1-2^{-53})$ se puede representar en *double precision*, sin embargo es necesario analizar en mayor detalle $\texttt{fl}(1-2^{-54})$. Particularmente se presenta la resta hasta el bit 53 de la mantisa de la siguiente forma:

  ```
  0 01111111110 1111111111111111111111111111111111111111111111111111|1.
  ```

  Esto significa que estamos tratando de almacenar el número que está justo en el punto medio entre $\texttt{fl}(1-2^{-53})$ y 1.0. En este caso la regla de redondeo nos indica que hay un `1` en el bit 53 y el resto de bits conocidos es `0` entonces se debe truncar a menos que el bit 52 sea `1`, lo que es justamente lo que ocurre en este caso, entonces se agrega `1` al bit 52. Al agregar `1` al bit 52 se obtiene 1.0.
- Entonces el menor valor de $m \in \mathbb{N}$ es 54.

(b) **[15 puntos]** Considere la siguiente expresión,

$$\xi(x) = \prod_{k=0}^{\infty} (2 - f_k(x)) = \prod_{k=0}^{\infty} \left(2 - \frac{1 - 2^{-2^k x}}{1 + 2^{-2^k x}}\right),$$

donde $\frac{1}{2^2} \leq x \leq 2^2$. Por conveniencia, aplicaremos la función $\log(x)$, que corresponde al logaritmo natural, entonces,

$$\log(\xi(x)) = \sum_{k=0}^{\infty} \log\left(2 - \frac{1 - 2^{-2^k x}}{1 + 2^{-2^k x}}\right).$$

Pregunta: Proponga un algoritmo que permita obtener el valor numérico de $\log(\xi(x))$, para un valor de x conocido, que asegure que se están sumando de forma **adecuada** todos los términos que **aportan** a la serie considerando que la computación se realizará en *double precision*.

Se sugiere:

- Argumentar claramente cómo evaluará cada término de la serie y cuantos términos usará.
- Justificar técnicamente cada una de las aproximaciones que realice.
- Respuesta que fundamente todos los puntos claves reciben puntaje completo.
- Considere que tiene a su disposición el uso de las funciones $\exp(x)$ y $\log(x)$.

Respuesta:

Análisis de los antecedentes:

- Se debe evaluar $\xi(x)$, lo que corresponde a una serie.
- También se sabe que $\log(1) = 0$
- La función $\log(x)$ es monótonamente creciente.

Argumentación técnica asociada al problema:

- La expresión $\left(\frac{1 - 2^{-2^k x}}{1 + 2^{-2^k x}}\right)$ es monótonamente creciente a 1 considerando valores positivos de x y $k \in \mathbb{N}_0$.
- La expresión $\left(2 - \frac{1 - 2^{-2^k x}}{1 + 2^{-2^k x}}\right)$ es monótonamente decreciente a 1 considerando valores positivos de x y $k \in \mathbb{N}_0$.
- De la respuesta anterior se sabe que $\mathtt{fl}\left(2 - \mathtt{fl}\left(\frac{1 - 2^{-54}}{1 + 2^{-54}}\right)\right) = 1$.
- **[5 puntos]** En *double precision* tiene sentido solo sumar los términos de la serie que son distintos a 0, en este caso esto se obtiene cuando $2^{\widehat{k}}\, x = 54$, despejando se obtiene,

  $$\begin{aligned} 2^{\widehat{k}}\, x &= 54, \\ 2^{\widehat{k}} &= \frac{54}{x}, \\ \widehat{k} &= \log_2\left(\frac{54}{x}\right). \end{aligned}$$

  Por lo tanto, el límite superior de la sumatoria debe ser $\left\lceil \widehat{k} \right\rceil$
- **[5 puntos]** Entonces se obtiene la siguiente sumatoria,

  $$\mathtt{fl}(\log(\xi(x))) = \sum_{k=0}^{\lceil \widehat{k} \rceil} \mathtt{fl}\left(\log\left(2 - \frac{1 - 2^{-2^k x}}{1 + 2^{-2^k x}}\right)\right).$$

  En este caso se pueden omitir términos dado que $\log(1) = 0$.
- Notar que el término $\left(2 - \frac{1 - 2^{-2^k x}}{1 + 2^{-2^k x}}\right)$ se puede manipular aún más, sin embargo como no hay errores de cancelación catastróficos se mantiene como está.

- **[5 puntos]** Por último, dado que $\left(2 - \dfrac{1 - 2^{-2^k x}}{1 + 2^{-2^k x}}\right)$ es monótonamente decreciente y la función $\log(x)$ preserva esta propiedad, tenemos que $\log\left(2 - \dfrac{1 - 2^{-2^k x}}{1 + 2^{-2^k x}}\right)$ es monótonamente decreciente. Por lo tanto para reducir el efecto de la pérdida de importancia en la sumatoria se procederá a sumar de "derecha" a "izquierda", esto significa que se debe sumar desde $k = \widehat{k}$ hasta $k = 0$, es decir de forma decreciente en k. Al sumar de "derecha" a "izquierda" se logra representar primero los números más pequeños e ir sumándolos, en caso contrario, la contribución de los números más pequeños se pierde.

(c) **[20 puntos]** Implemente en Python utilizando adecuadamente la librería NumPy (en especial su capacidad de vectorización) el algoritmo propuesto en la Pregunta 1 ítem (b), es decir, en la pregunta anterior. Para su implementación, considere que **solo** tiene a su disposición las siguientes funciones de la librería NumPy, además de las operaciones elementales, ciclos y condicionales propios de Python:

- `np.arange(n)`: Para `n` un número entero positivo entrega un vector de largo `n` con números enteros desde `0` a `n-1`.
- `np.arange(start, stop, step)`: Genera un arreglo de NumPy semi-abierto con los valores `[start, stop[` con espaciado entre elementos igual a `step`. Por ejemplo: `np.arange(2, 5, 1)=[2, 3, 4]` o `np.arange(10, 5, -1)=[10, 9, 8, 7, 6]`.
- `np.abs(x)`: Entrega el valor absoluto de `x`.
- `np.power(x,n)`: Evalúa la expresión x^n si `x` y `n` son escalares. En caso de que `x` e `n` sean vectores, deben tener la misma dimensión y entrega la evaluación elemento a elemento. Si solo uno de los términos es un vector, entrega el vector donde el término constante se consideró para cada término de vector.
- `np.sqrt(x)`: Entrega la evaluación de la raíz cuadrada no negativa de un vector o escalar `x`.
- `np.exp(x)`: Entrega la evaluación de la función exponencial de un vector o escalar `x`.
- `np.log(x)`: Entrega la evaluación de la función logaritmo natural de un vector o escalar `x` positivo.
- `np.ceil(x)`: Entrega la parte entera superior de cada elemento del vector o escalar `x`.
- `np.floor(x)`: Entrega la parte entera interior de cada elemento del vector o escalar `x`.

Notar que al momento de implementar usted **debe decidir** qué componentes se deben vectorizar y qué componentes no, considerando las funciones de NumPy antes mencionadas. Considere la siguiente firma:

```
'''
input:
x       : (float) Input value for the approximation of log(xi(x)).

output:
lxi     : (float) The computed values for log(xi(x)).
'''
def compute_log_xi(x):
    # Your own code.
```

```
    return lxi
```

Respuesta:

```
'''
input:
x       : (float) Input value for the approximation of log(xi(x)).

output:
lxi     : (float) The computed values for log(xi(x)).
'''
def compute_log_xi(x):
```

- **[5 puntos]** Definir la función a evaluar.

```
    # Definition of auxiliary functions and the required function itself
    f_num = lambda k,x: (1.-np.power(2.,-np.power(2.,k)*x))
    f_den = lambda k,x: (1.+np.power(2.,-np.power(2.,k)*x))
    g = lambda k,x: np.log(2.-f_num(k,x)/f_den(k,x))
```

- **[5 puntos]** Determinar límite superior a considerar de la sumatoria.

```
    # Given that 2^{-2} <= x <= 2^{2}, we will not get x>54, which implies that 0<log(54/x).
    # This solves the problem of getting k_hat negative.
    k_hat = np.ceil(np.log(54./x)/np.log(2))
```

- **[10 puntos]** Sumar adecuadamente los términos, esto requiere haber definido los términos de forma adecuada previamente.

```
    # Array in inverse order from k_hat until 0.
    ks = np.arange(k_hat,-1,-1)

    # Adding from "right" to "left"
    lxi = 0.0
    for k in ks:
        lxi += g(k,x)

    # Returning the output
    return lxi
```