

Trabajo Práctico 1

Aprendizaje Automático Avanzado

Cisnero Matias, Seivane Nicolás, Serafini Franco
22 de Septiembre de 2025

Ejercicio 1: Creación de Corpus



1.1 Descripción de Librerías Usadas

Se utilizaron las librerías de *r* y *pdfplumber*, en la cual utilizamos la ultima para leer página por página de un pdf y la primera para seleccionar las palabras.

Los links a las librerías son los siguientes.

pdfplumber

r

Funciones Utilizadas

- `pdfplumber.open()` as
pdf
- `pdf.pages[]`
- `.extract_text()`
- `.split('\n')`
- `re.findall()`
- `.endswith()`
- `.strip()`
- `.isdigit()`
- `.split('\n')`

1.2.1 Estructura de Código

Se utiliza la siguiente estructura de código:

Se comienza importando las librerías y creando una lista de palabras, donde se irán agregando las extracciones de texto.

```
import pdfplumber
import re

words = []
```

En lo cual se sigue utilizando la función `pdfplumber.open()` as `pdf`, en la cual se debe especificar la ruta hacia el pdf. El cual nos devuelve *pdf* como una instancia de la clase `pdfplumber.PDF`

```
with pdfplumber.open("ruta") as pdf:
```

1.2.2 Estructura de Código

Se continua utilizando una propiedad de la clase `pdfplumber.Page`, de la cual se puede indexar para acceder a las paginas del pdf

```
with pdfplumber.open("ruta") as pdf:
    for page in pdf.pages[:]:
```

En lo cual se utiliza el metodo `.extract_text()`, que recopila todos los objetos de caracteres de la página en un solo string.

```
with pdfplumber.open("ruta") as pdf:
    for page in pdf.pages[:]:
        text = page.extract_text()
        if text:
```

1.2.3 Estructura de Código

Se continua diviendo el string segun el metodo `.split('\n')`, el cual devuelve una lista de strings, los cuales fueron separados de acuerdo a `\n`, ergo saltos de linea.

```
with pdfplumber.open("ruta") as pdf:
    for page in pdf.pages[:]:
        text = page.extract_text()
        if text:
            lines = text.split('\n')
```

Luego se sacan las lineas que sean numeros de pagina tanto en el pie de la misma como en el encabezado. La forma de extraccion varia de acuerdo a como es el pdf.

```
if lines[-1].strip().isdigit():
    lines = lines[:-1]
if lines[0].strip().isdigit():
    lines = lines[1:]
```

1.2.4 Estructura de Código

Se crea por linea una lista con el método de la librería r:

`re.findall(r"_w+|[,.!?:;]", line)`, en el cual se separan con expresiones regulares las palabras con `\w+` y aparte los signos de puntuación con `[,.!?:;]`, en una lista de strings. Luego para cada palabra se la pasa a minúscula con el método `.lower()`.

```
with pdfplumber.open("ruta") as pdf:
    for page in pdf.pages[:]:
        text = page.extract_text()
        if text:
            lines = text.split('\n')
            if lines[-1].strip().isdigit():
                lines = lines[:-1]
            if lines[0].strip().isdigit():
                lines = lines[1:]
            for line in lines:
                tokens = re.findall(r"\w+|[,.!?:;]", line)
                tokens = [token.lower() for token in tokens]
```

1.2.5 Estructura de Código

Luego se diferencia por línea los puntos aparte, los cuales consideramos los últimos puntos de las líneas. Cada línea, las cuales fueron convertidas en listas de strings son agregadas a la lista del corpus

```
with pdfplumber.open("ruta") as pdf:
    for page in pdf.pages[:]:
        text = page.extract_text()
        if text:
            lines = text.split('\n')
            if lines[-1].strip().isdigit():
                lines = lines[:-1]
            if lines[0].strip().isdigit():
                lines = lines[1:]
            for line in lines:
                tokens = re.findall(r"\_w+|[\.,!?:;]", line)
                tokens = [token.lower() for token in tokens]
            if line.endswith("."):
                tokens[-1] = "."
            words.extend(tokens)
```

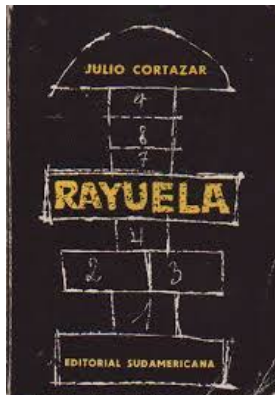

1.3 Libros utilizados: Rayuela

Título: Rayuela

Autor: Julio Cortazar

Año : 1963

Se extrayeron 197.342 caracteres y 20.810 caracteres únicos que conforman el vocabulario.



1.3.2 Código utilizado: Rayuela

```
with pdfplumber.open("Julio-Cortazar-Rayuela.pdf") as pdf:
    for page in pdf.pages[7:]:
        text = page.extract_text()
        if text:
            lines = text.split('\n')
            if lines[-1].strip().isdigit():
                lines = lines[:-1]
            if lines[0].strip().isdigit():
                lines = lines[1:]
            if lines[0].strip().isdigit():
                lines = lines[1:]
            if lines[-1].strip().isdigit():
                lines = lines[:-1]
            for line in lines:
                tokens = re.findall(r"\w+|[\.,!?:;]", line)
                tokens = [token.lower() for token in tokens]
            if line.endswith("."):
                tokens[-1] = ". "
            words.extend(tokens)
```

1.3.3 Ejemplo Borrado: Rayuela

ganar de reir, el miedo me hacía una doble llave en la boca del estómago y al final me dio una verdadera desesperación (el mozo se había levantado furioso) y empecé a agarrar los zapatos de las mujeres y a mirar al debajo del arco de la suela no estaría agasapado el azúcar, y las gallinas cacareaban, los gallos gerentes me picoteaban el lomo, oía las carcajadas de Ronald y de Kienne mientras me movía de una mesa a otra hasta encontrar el azúcar escondido detrás de una pata Segundo Imperio. Y todo el mundo enfurecido, hasta yo con el azúcar apretado en la palma de la mano y sintiendo cómo se mezclaba con el sudor de la piel, cómo asquerosamente se deshacía en una especie de venganza pegajosa, esa clase de episodios todos los días.

4º Caso de borrado (-2)

1º Caso de borrado 9

2º Caso de borrado 2

3º Caso de borrado

Aquí había sido primero como una sangría, un vapuleo de uso interno, una necesidad de sentir el estúpido pasaporte de tapas azules en el bolsillo del saco, la llave del hotel bien segura en el clavo del tablero. El miedo, la ignorancia, el deslumbramiento: Esto se llama así, eso se pide así, ahora esa mujer va a sonreír, más allá de esa calle empieza el Jardín des Plantes. París, una tarjeta postal con un dibujo de Kien al lado de un espejo vacío.

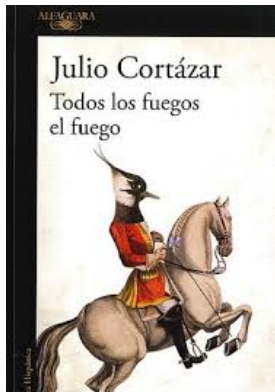
1.3 Libros utilizados: Todos los fuegos

Título: Todos los fuegos el fuego.

Autor: Julio Cortázar

Año : 1966

Se extrayeron 55.948 caracteres y 2.828 caracteres únicos que conforman el vocabulario.



1.3.2 Código utilizado: Todos los fuegos

```
with pdfplumber.open("Julio Cortazar Todos los fuegos.pdf")
    as pdf:
    for page in pdf.pages[:-1]:
        text = page.extract_text()
        if text:
            lines = text.split('\n')
            if lines[-1].strip().isdigit():
                lines = lines[:-1]
            for line in lines:
                tokens = re.findall(r"\w+|[\.,!?:;]", line)
                tokens = [token.lower() for token in tokens]
            if line.endswith("."):
                tokens[-1] = "."
            words.extend(tokens)
```

1.3.3 Ejemplo Borrado: Todos los fuegos

A la cuarta vez de encontrarse con todo eso, de hacer todo eso, el ingeniero había decidido no salir más de su coche, a la espera de que la policía disolviese de alguna manera el embotellamiento. El calor de agosto se sumaba a ese tiempo a ras de neumáticos para que la inmovilidad fuese cada vez más enervante. Todo era olor a gasolina, gritos destemplados de los jovencitos del Simca, brillo del sol rebotando en los cristales y en los bordes cromados, y para colmo la sensación contradictoria del encierro en plena selva de máquinas pensadas para correr. El 404 del ingeniero ocupaba el segundo lugar de la pista de la derecha contando desde la franja divisoria de las dos pistas, con lo cual tenía otros cuatro autos a su derecha y siete a su izquierda, aunque de hecho sólo pudiera ver distintamente los ocho coches que lo rodeaban y sus ocupantes que ya había detallado hasta cansarse. Había charlado con todos, salvo con los muchachos del Simca que le caían antipáticos; entre trecho y trecho se había discutido la situación en sus menores detalles, y la impresión general era que hasta Corbeil-Essones se avanzaría al paso o poco menos, pero que entre Corbeil y Juvisy el ritmo iría acelerándose una vez que los helicópteros y los motociclistas logaran quebrar lo peor del embotellamiento. A nadie le cabía duda de que algún accidente muy grave debía haberse producido en la zona, única explicación de una lentitud tan increíble. Y con eso el gobierno, el calor, los impuestos, la vialidad, un tópico tras otro, tres metros, otro lugar común, cinco metros, una frase sentenciosa o una maldición contenida.

A las dos monjitas del 2HP les hubiera convenido tanto llegar a Milly-la-Forêt antes de las ocho, pues llevaban una cesta de hortalizas para la cocinera. Al matrimonio del Peugeot 203 le importaba sobre todo no perder los juegos televisados de las nueve y media; la muchacha del Dauphine le había dicho al ingeniero que le daba lo mismo llegar más tarde a París pero que se quejaba por principio, porque le parecía un atropello someter a millares de personas a

Único Caso Borrado 3

un régimen de caravana de camellos. En esas últimas horas (debían ser casi las cinco pero el calor los hostigaba insoportablemente) habían avanzado unos cincuenta metros a juicio del ingeniero, aunque uno de los hombres del Taunus que se había acercado a charlar llevando de la mano al niño con su autito, mostró irónicamente la copa de un plátano solitario y la

1.3 Libros utilizados: Historias de cronopios y de famas

Título: Historias de cronopios y de famas.

Autor: Julio Cortazar

Año : 1962

Se extrayeron 32.224 caracteres y 2.514 caracteres únicos que conforman el vocabulario.



1.3.2 Código utilizado: Historias de cronopios y de famas

En este caso no fue necesario quitar ninguna linea.

```
with pdfplumber.open("Historias-de-Cronopios-y-de-Famas -
    Julio Cortazar.pdf") as pdf:
    for page in pdf.pages[3:-1]:
        text = page.extract_text()
        if text:
            lines = text.split('\n')
            for line in lines:
                tokens = re.findall(r"\w+|[\.,!?:;]", line)
                tokens = [token.lower() for token in tokens]
            if line.endswith("."):
                tokens[-1] = ". "
            words.extend(tokens)
```

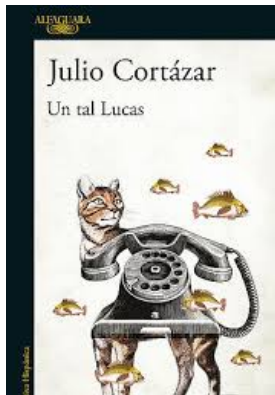

1.3 Libros utilizados: Un tal Lucas.

Título: Un tal Lucas.

Autor: Julio Cortazar

Año : 1979

Se extrayeron 32.224 caracteres y 2.514 caracteres únicos que conforman el vocabulario.



1.3.2 Código utilizado: Un tal Lucas.

```
with pdfplumber.open("Lucas_Julio_Cortazar.pdf") as pdf:
    for page in pdf.pages[5:]:
        text = page.extract_text()
        if text:
            lines = text.split('\n')
            if lines[-1].strip().isdigit():
                lines = lines[:-1]
            lines = lines[1:]
            for line in lines:
                tokens = re.findall(r"\w+|[\.,!?:;]", line)
                tokens = [token.lower() for token in tokens]
            if line.endswith("."):
                tokens[-1] = "."
            words.extend(tokens)
```

1.3.3 Ejemplo Borrado: Un tal Lucas.

2º Caso de Borrado **Un Tal Lucas – Julio Cortázar**

Lucas, su patriotismo

De mi pasaporte me gustan las páginas de las renovaciones y los sellos de visados redondos / triangulares / verdes / cuadrados / negros / ovalados / rojos, de mi imagen de Buenos Aires el transbordador sobre el Riachuelo, la plaza Irlanda, los jardines de Agronomía, algunos cafés que acaso ya no están, una cama en un departamento de Maipú casi esquina Córdoba, el olor y el silencio del puerto a medianoche en verano, los árboles de la plaza Lavalle.

Del país me queda un olor de acequias mendocinas, los álamos de Uspallata, el violeta profundo del cerro de Velasco en La Rioja, las estrellas chaqueñas en Pampa de Guanacos yendo de Salta a Misiones en un tren del año cuarenta y dos, un caballo que monté en Saladillo, el sabor del Cinzano con ginebra Gordon en el Boston de Florida, el olor ligeramente alérgico de las plateas del Colón, el superpúlman del Luna Park con Carlos Beulchi y Mario Díaz, algunas lecherías de la madrugada, la fealdad de la Plaza Once, la lectura de *Sur* en los años dulcemente ingenuos, las ediciones a cincuenta centavos de *Claridad*, con Roberto Arlt y Castelnuovo, y también algunos patios, claro, y sombras que me callo, y muertos.

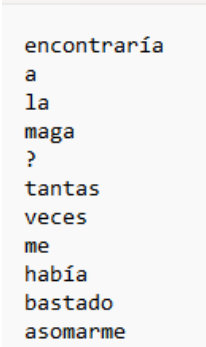
1º Caso de Borrado **10**

1.4 Corpus final

Se guarda el corpus final en un archivo llamado **corpus.txt**.

```
with open("corpus.txt", "w", encoding="utf-8") as f:  
    f.write("\n".join(words))
```

Obteniendo un corpus como se ve en la siguiente imagen.

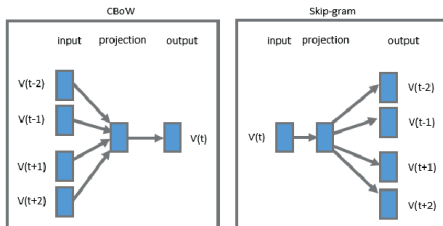


```
encontraría  
a  
la  
maga  
?  
tantas  
veces  
me  
había  
bastado  
asomarme
```

Vocabulario (único): 27.971

Corpus total: 310.347

Ejercicio 2: Implementación CBOW y SkipGram



2.1 Pasos previos

Se abre y carga el **corpus.txt**.

```
with open("corpus.txt", "r", encoding="utf-8") as f:  
    corpus = f.read().splitlines()
```

Luego se crean los diccionarios que se utilizaran en ambos métodos.

```
vocab = sorted(set(corpus))  
vocab_tamano = len(vocab)  
palabra_a_indice = {palabra: i for i, palabra in  
enumerate(vocab)}  
indice_a_palabra = {i: palabra for i, palabra in  
enumerate(vocab)}
```

2.2 CBOW: Conceptos Fundamentales

Definición: Continuous Bag of Words(CBOW)

Propósito: El objetivo es encontrar representaciones vectoriales (v_p) para cada palabra en el vocabulario V . Estas representaciones se optimizan para que las palabras que comparten contextos sean más similares (tengan un producto interno alto)

Principio: A diferencia de los modelos más simples, CBOW utiliza un contexto de C palabras para predecir una palabra central

Contexto vs. Predicción: A partir de un contexto de C palabras ($p_{I,1}, p_{I,2}, \dots, p_{I,C}$), se intenta predecir la palabra objetivo (p_O), que generalmente es la palabra central

2.2 CBOW: Arquitectura del Perceptrón

Arquitectura

Estructura: El calculo se enmarca en un **perceptron multicapa**

Entrada: Se presentan las C palabras de contexto. La representación de las entradas se realiza mediante la **codificación One-hot**

Diccionario (V): Las palabras pertenecen a un diccionario V cuyo cardinal es $|V|$

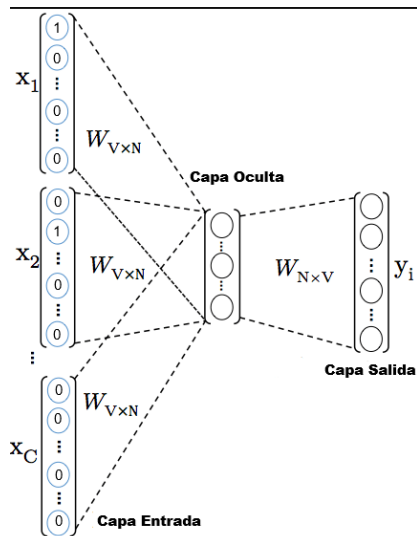
Capa Oculta: Tiene una única capa oculta con N unidades, todas con función de activación lineal $g(x) = x$

Dimensiones de Pesos:

- Matriz de pesos Entrada-Oculta (W): $W \in |V| \times N$
- Matriz de pesos Oculta-Salida (W'): $W' \in N \times |V|$

Salida: La función de activación de las unidades de salida es soft-max

2.2 CBOW: Arquitectura del Perceptrón



2.2 Calculo de representación Contextual (h)

Cálculo de h : Se h calcula como el **promedio de las filas** de W que corresponden a los índices de las palabras en el contexto

Representaciones One-Hot: Sea x_1, x_2, \dots, x_C las representaciones one-hot de las C palabras en el contexto

Representaciones Contextuales (v_p): Sea v_{p_i} la representación contextual de la palabra p_i (que es la fila i de W donde i es el índice de p_i en V)

Fórmula de h :

- Cálculo del vector h :

$$h = \frac{1}{C} W^t (x_1 + x_2 + \dots + x_C)$$

- Cálculo de h usando v_p :

$$h = \frac{1}{C} (v_{p1} + v_{p2} + \dots + v_{pC})$$

2.2 Calculo de representación Contextual (h)

```
h = cp.mean(W[indices_contextos], axis=0).reshape(-1,1)
```

Donde se utiliza la función `cupy.mean(a, axis=None, dtype=None, out=None, keepdims=False)`, que retorna la media de la matriz de entrada a a lo largo del eje.

Explicación

- `W[indices_contextos]` selecciona los vectores de embedding correspondientes a esas palabras
- `axis=0` significa promediar columna por columna.

$$h = \frac{1}{C} \sum_{i=1}^C v_{p_i} = \left(\frac{1}{C} \sum_{i=1}^C w_{i1}, \frac{1}{C} \sum_{i=1}^C w_{i2}, \dots, \frac{1}{C} \sum_{i=1}^C w_{id} \right)$$

2.2 Salida de la Red

Excitación de la salida

Vector de Salida (v'_{p_j}): La columna j de la matriz W' se denota como v'_{p_j} , siendo el vector de salida para la palabra p_j

Estado de Excitación (u_j) El estado de excitación de cada unidad de salida j (u_j) se calcula como el producto interno del vector de salida v'_{p_j} y la activación de la capa oculta h

$$u_j = (\mathbf{v}'_{p_j})^t \mathbf{h}$$

Siendo en el código

```
u = W_prima.T@h
```

2.2 Salida de la Red (Softmax)

Excitación de la salida

Activación de Salida (y_j): La función de activación de las unidades de salida es soft-max. El estado de activación y_j de la unidad j se calcula de la siguiente manera

$$y_j = \frac{\exp u_j}{\sum_{j'=1}^{|V|} \exp u_{j'}}$$

Interpretación: El valor de y_j se interpreta como la probabilidad de que la palabra objetivo sea p_j dado el contexto que entra a la red $(p_{I,1}, \dots, p_{I,C})$

$$P(p_j/p_{I,1}, \dots, p_{I,C}) = y_j$$

Siendo en el código

```
y = softmax(u)
```

2.2 Salida de la Red (Softmax)

Excitación de la salida

Siendo en el código

```
def softmax(u):
    u_max = np.max(u)
    e_u = np.exp(u - u_max)
    return e_u / e_u.sum()
```

Consideraciones: Se le resta a u el máximo de u , ya que generaba errores *nan* en los pesos

Explicación: Restar u_max evita problemas numéricos por exponentes grandes, llamado overflow

$$\frac{e^{u_i - c}}{\sum_j e^{u_j - c}} = \frac{e^{u_i} e^{-c}}{\sum_j e^{u_j} e^{-c}} = \frac{e^{u_i}}{\sum_j e^{u_j}}$$

2.2 Función de Pérdida (E)

Objetivo: Minimizar la función de pérdida E , que es el negativo del logaritmo de la probabilidad de la palabra objetivo p_O dado el contexto $(p_{I,1}, \dots, p_{I,C})$.

Pérdida Logarítmica:

$$E = -\log p(p_O \mid p_{I,1}, p_{I,2}, \dots, p_{I,C})$$

Error (e_j): Se define como la diferencia entre la predicción y el objetivo:

$$e_j = y_j - t_j$$

donde

$$t_j = \begin{cases} 1 & \text{si } j = j^* \text{ (índice de la palabra objetivo } p_O) \\ 0 & \text{si } j \neq j^* \end{cases}$$

2.2 Función de Pérdida y Derivada

Función de Pérdida

La función de pérdida se define como:

$$\begin{aligned} E &= -\log p(p_O \mid p_{I,1}, p_{I,2}, \dots, p_{I,C}) \\ &= -u_{j^*} + \log \sum_{j'=1}^{|V|} \exp u_{j'} \end{aligned}$$

Derivada del error respecto al estado de excitación u_j :

$$\frac{\partial E}{\partial u_j} = y_j - t_j = e_j$$

En código es

```
e = y
e[indice_central] -= 1
```


2.2 Actualización de W' (Pesos Oculta-Salida)

Actualización de pesos

La regla de actualización para W' en CBOW es idéntica a la del contexto de una sola palabra, ya que solo se modificó el cálculo de h

Derivando E respecto a w'_{ij} obtenemos la regla de actualización:

En forma vectorial:

$$W'_{:,j}(\text{nuevo}) = W'_{:,j}(\text{anterior}) - \eta e_j h$$

Considerando las columnas de W' como vectores de salida v'_j :

$$v'_j(\text{nuevo}) = v'_j(\text{anterior}) - \eta e_j h$$

En código es

```
W_prima -= n*(h@e.T)
```

2.2 Vector de Error Propagado (EH):

Error Propagado

Este vector de error se propaga hacia la capa oculta y se calcula como el producto de la matriz W' por el vector de errores de salida e

$$EH = W'e$$

La regla de actualización para W se aplica a la fila W_{lc} de W (o su vector de representación contextual v_{l_c}) para cada palabra l_c en el contexto, donde $1 \leq c \leq C$

$$v_{l_c}(\text{nuevo}) = v_{l_c}(\text{anterior}) - \eta \frac{1}{C} EH^t$$

En código es

```
EH = W_prima@e
W[indices_contextos] -= n * EH.T / len(
    indices_contextos)
```

2.2 Código Entero

Código Python

```
for epoca in range(epocas):
    for i, (indice_central, indices_contextos) in enumerate(
        indices_tuplas):
        h = cp.mean(W[indices_contextos], axis=0).reshape(
            (-1,1))
        u = W_prima.T @ h
        y = softmax(u)

        e = y
        e[indice_central] -= 1
        W_prima -= n * (h @ e.T)
        EH = W_prima @ e
        W[indices_contextos] -= n * EH.T / len(
            indices_contextos)
```

2.3 Introducción al Muestreo Negativo (NS)

Muestreo Negativo: Reducción de la Complejidad

Problema Principal:

- En el entrenamiento estándar (usando Softmax en la capa de salida), la regla de actualización de pesos debe aplicarse a **cada palabra** del vocabulario V .
- La matriz de pesos de salida W' debe ser actualizada para las $|V|$ palabras.
- Cuando el tamaño del vocabulario $|V|$ es muy grande, el **costo computacional** del proceso se incrementa significativamente.

Solución: Muestreo Negativo (NS):

- El muestreo negativo aborda este problema definiendo un **subconjunto pequeño** de palabras P_{sel} .
- La actualización de los pesos W' y W se realiza considerando **solo la palabra objetivo** (p_O) y los ejemplos negativos en P_{sel} .
- El tamaño de P_{sel} (casos negativos) es considerablemente menor que $|V|$.

2.3 La Nueva Función de Pérdida (Loss Function)

Cambio de Arquitectura: Reducción de la Complejidad

- Para evitar la costosa normalización Softmax (que requiere la suma sobre $|V|$ elementos), NS modifica la función de pérdida para utilizar la **función sigmoidea logística** $\sigma(x) = \frac{1}{1+e^{-x}}$.

Definición del Conjunto de Muestras:

- El entrenamiento se basa en el conjunto $p_O \cup P_{sel}$, donde p_O es el ejemplo positivo (la palabra objetivo a predecir) y P_{sel} es el conjunto de ejemplos negativos, que **no debe incluir a p_O** .

Fórmula de la Función de Pérdida (E):

- La función de pérdida negativa está definida como:

$$E = -\log \sigma((v' p_O)^t h) - \sum_{p_j \in P_{sel}} \log \sigma(-(v'_{p_j})^t h)$$

Objetivo de Minimización:

- El término $-\log \sigma((v' p_O)^t h)$ busca **maximizar** la probabilidad de que p_O sea la palabra objetivo (ejemplo positivo).

2.3 Reglas de Actualización de Pesos

- El error e_j para una unidad de salida j ya no se calcula con Softmax, sino utilizando la sigmoide y el valor deseado t_j (donde $u_j = (v'_{p_j})^t h$).

$$e_j = \sigma(u_j) - t_j$$

.

- Si p_j es la palabra objetivo (p_O), entonces $t_j = 1$ y $e_O = \sigma(u_O) - 1$.
- Si p_j es un ejemplo negativo ($p_j \in P_{sel}$), entonces $t_j = 0$ y $e_j = \sigma(u_j)$.

Actualización de W' (Vectores de Salida v'_j):

- La actualización solo se aplica a los vectores v'_j para $j \in p_O \cup P_{sel}$.

$$v'_j(\text{nuevo}) = v'_j(\text{anterior}) - \eta e_j h$$

Donde η es la tasa de aprendizaje ($\eta > 0$).

2.3 Reglas de Actualización de Pesos

Actualización de W (Vectores de Contexto v_I):

- Para actualizar la matriz de entrada W , primero necesitamos calcular el vector de error retropropagado EH .
- EH se calcula como la suma ponderada de los vectores de salida v'_j por el error e_j , pero solo para el conjunto de muestras $p_O \cup P_{sel}$:

$$EH = \sum_{p_j \in p_O \cup P_{sel}} e_j v'_{p_j}$$

- Una vez calculado EH , se actualizan los vectores de entrada (v_I) que forman el contexto. (Para el caso CBOW, la actualización vectorial de la fila I_c en W correspondiente a cada palabra de contexto I_c debe incluir el factor de promedio $\frac{1}{C}$, como se hacía en la formulación Softmax estándar de CBOW).

2.3 Negativos CBOW en código

Código Python

```
for epoca in range(epocas):
    for i, (indice_central, contexto_idx, negativos_idx) in
        enumerate(indices_tuplas):

        h = cp.mean(W[contexto_idx], axis=0).reshape(-1, 1)
        subconjunto = list(set([indice_central] +
                                negativos_idx))
        u_sub = W_prima[:, subconjunto].T @ h

        y = sigmoid(u_sub)
        EL_sub = y
        pos_idx = subconjunto.index(indice_central)
        EL_sub[pos_idx] -= 1

        W_prima[:, subconjunto] -= n * (h @ EL_sub.T)
        EH = W_prima[:, subconjunto] @ EL_sub
        W[contexto_idx] -= n * EH.T / len(contexto_idx)
```


2.3 Generación de tuplas Central + Contexto

Objetivo: Preparar los datos para CBOW o Skip-gram creando tuplas de la forma:

(índices palabra central, índices palabras de contexto)

- Para cada palabra del corpus, se considera una **ventana de contexto** con tamaño definido.
- Las palabras dentro de la ventana forman el **contexto** de la palabra central.
- Cada tupla representa un ejemplo de entrenamiento:
 - **Palabra central:** la que queremos predecir o usar como input.
 - **Contexto:** palabras vecinas que proveen información semántica.
- Este paso es a los algoritmos, para no utilizar memoria extra en las iteraciones del código.

Ejemplo conceptual: Corpus: [“el”, “gato”, “come”, “pescado”] Ventana de contexto = 1 → Tuplas generadas:

(“gato”, [“el”, “come”]), (“come”, [“gato”, “pescado”])

2.3 Selección de Tuplas - Código

```
def generar_tuplas(corpus, palabras_a_indice, contexto):  
  
    indices = [i for i in range(contexto, (len(corpus)-contexto  
        ))]  
  
    indices_contexto = [i for i in range(-contexto,0)] + [i  
        for i in range(1,contexto+1)]  
  
    indices_tuplas = []  
  
    for i in indices:  
  
        indices_tuplas.append((palabras_a_indice[corpus[i]], [  
            palabras_a_indice[corpus[i+j]] for j in indices_contexto  
        ]))  
  
    return indices_tuplas
```

2.3 Selección de Ejemplos Negativos

Objetivo: Seleccionar palabras negativas para Skip-gram / CBOW.

- Para cada palabra central, tomamos su **contexto real** (palabras cercanas).
- El conjunto de **negativos** se obtiene de palabras fuera del contexto.
- Esto permite que el modelo aprenda a diferenciar **palabras correctas vs. negativas** sin actualizar todo el vocabulario.
- La selección puede ser:
 - **Determinística:** alrededor de la posición de la palabra central.
 - **Aleatoria:** muestreo uniforme de vocabulario excluyendo contexto.

2.3 Selección de Ejemplos Negativos

- **Negativos determinísticos:** se eligen palabras cercanas a la central, pero fuera de la ventana de contexto. Ejemplo: palabra central = come, ventana = 1
Contexto: [gato, pescado]
Negativos determinísticos: [e1] (palabra izquierda cercana, fuera de contexto)
- **Negativos aleatorios:** se seleccionan de forma uniforme de todo el vocabulario, excluyendo la central y el contexto.
Ejemplo: vocabulario = [e1, gato, come, pescado, pez]
Contexto: [gato, pescado]
Negativos aleatorios: [e1, pez] (cualquier palabra fuera del contexto)

2.3 Selección de Negativos - Código Determinístico

Ejemplos conceptuales:

```
def generar_tuplas_con_negativos(corpus, palabras_a_indice,
    contexto, num_negativos):
    indices = [i for i in range(contexto, len(corpus)-
contexto)]
    indices_contexto = [i for i in range(-contexto,0)] + [
i for i in range(1,contexto+1)]
    indices_tuplas = []
    for i in indices:
        indices_tuplas.append(
            (palabras_a_indice[corpus[i]],
            [palabras_a_indice[corpus[i + j]] for j in
indices_contexto],
            obtener_negativas(corpus, i, contexto, num_negativos
)))
    return indices_tuplas
```

2.3 Selección de Negativos - Código Determinístico

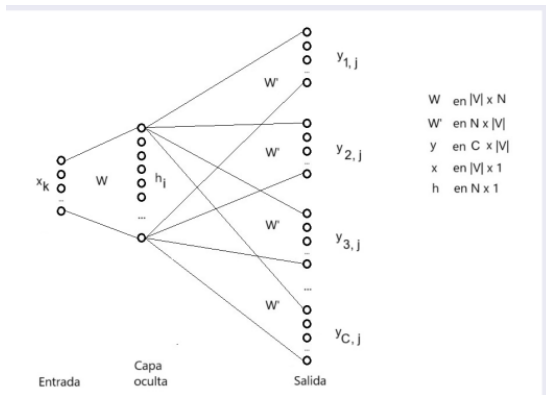
```
def obtener_negativas(corpus, indice, contexto,
num_negativos=5):
    maximo = contexto + num_negativos // 2
    inicio = max(0, indice - maximo)
    fin = min(len(corpus), indice + maximo + 1)
    izq = corpus[max(0, inicio - maximo):inicio]
    der = corpus[fin:min(len(corpus), fin + maximo)]
    if len(izq) < num_negativos // 2:
        faltan = (num_negativos // 2) - len(izq)
        der = corpus[fin:min(len(corpus), fin + maximo +
faltan)]
    elif len(der) < num_negativos // 2:
        faltan = (num_negativos // 2) - len(der)
        izq = corpus[max(0, inicio - maximo - faltan):inicio]
    candidatas = [p for p in izq + der if p != corpus[indice]]
    return [palabras_a_indice[p] for p in candidatas[:
num_negativos]]
```

2.3 Selección de Negativos - Código Aleatorio

```
def generar_tuplas_con_negativos_random(corpus,
    palabras_a_indice, contexto, num_negativos):
    indices = [i for i in range(contexto, len(corpus)-
contexto)]
    indices_contexto = [i for i in range(-contexto,0)] + [
i for i in range(1,contexto+1)]
    vocabulario = set(range(len(palabras_a_indice)))
    indices_tuplas = []
    for i in indices:
        indices_tuplas.append(
            (palabras_a_indice[corpus[i]],
            [palabras_a_indice[corpus[i + j]] for j in
indices_contexto],
            random.sample(
                vocabulario - set([palabras_a_indice[corpus[i + j]]
for j in indices_contexto]),
                k=num_negativos)))
    return indices_tuplas
```

2b.1.1 Skip-gram

Arquitectura de Skip-gram



2b.1.2 Skip-gram

Definición: Skip-gram

Propósito: Es un modelo de aprendizaje automático para aprender representaciones de palabras que capturan el significado de las palabras basadas en su contexto. La diferencia con CBOW es que la entrada de la red es una palabra y la salida intenta predecir su contexto.

Principio: Skip-gram utiliza una palabra central para predecir un contexto de C palabras.

Contexto vs. Predicción: A partir de una palabra central (p_O), el modelo intenta predecir las palabras de su contexto ($p_{I,1}, p_{I,2}, \dots, p_{I,C}$).

2b.1.3 Skip-gram: Representación de la entrada

Entrada del modelo

Palabra central: se representa como un vector one-hot $x \in \mathbb{R}^{|V|}$, donde $|V|$ es el tamaño del vocabulario.

Ejemplo: si $|V| = 10\,000$ y la palabra central ocupa la posición 22, entonces x tiene un 1 en la posición 22 y 0 en las demás.

Matriz de pesos de las conexiones entre la entrada y la capa oculta:

$$W \in \mathbb{R}^{|V| \times N}$$

donde N es la dimensión del espacio de las representaciones vectoriales (ej. $N = 300$).

2b.1.4 Skip-gram: Representación vectorial

Cálculo de la representación vectorial de la palabra p_I

La capa oculta no tiene función de activación. Se obtiene directamente la representación vectorial de la palabra central:

$$h = W^T x = v_{p_I}, \quad h \in \mathbb{R}^N$$

donde v_{p_I} es la representación vectorial asociado a la palabra central p_I .

Dimensiones:

- $x \in \mathbb{R}^{|V|}$ (one-hot).
- $W \in \mathbb{R}^{|V| \times N}$.
- $h \in \mathbb{R}^N$ (representación vectorial asociado a la palabra p_I).

2b.1.5 Skip-gram: Salida y softmax

Cálculo de probabilidades

Para cada palabra j en el vocabulario, se calcula:

$$u_j = (v'_j)^T h, \quad u \in \mathbb{R}^{|V|}$$

donde v'_j es el vector de salida correspondiente a la palabra j , y $W' \in \mathbb{R}^{N \times |V|}$ es la matriz de salida.

Luego se aplica softmax:

$$y_j = \frac{\exp(u_j)}{\sum_{k=1}^{|V|} \exp(u_k)}$$

obteniendo la probabilidad de que la palabra j aparezca en el contexto de p_I .

2b.1.6 Skip-gram: Función de pérdida

Pérdida logarítmica

Dado un contexto de C palabras alrededor de la palabra central p_I , la función de pérdida se define como:

$$E = - \sum_{c=1}^C \log P(p_{O_c} | p_I)$$

donde p_{O_c} son las palabras del contexto.

2b.1.7 Skip-gram: Actualización

Reglas de actualización

Durante el entrenamiento, los vectores de entrada y salida se ajustan.

Para los vectores de salida:

$$v'_j \leftarrow v'_j - \eta E_{Ij} h$$

Para el vector de entrada de la palabra central:

$$v_{p_I} \leftarrow v_{p_I} - \eta E H^T$$

donde:

- η es la tasa de aprendizaje.
- E_{Ij} depende del error para cada palabra del contexto.

2b.1.8 Skip-gram: Implementación completa en Python

```
def entrenar_skipgram(ruta_corpus, nombre_pc, epocas
=1, eta=0.001, N=300, C=4, W1=None, W2=None,
intervalo_guardado=50):

    corpus, vocab, vocab_size, word_to_idx, idx_to_word
= cargar_corpus(ruta_corpus)
    W1, W2 = inicializar_pesos(vocab_size, N, W1, W2,
cpararray=True)
    indice_tuplas = generar_tuplas_central_contexto(
corpus, word_to_idx, C)
    total_pares = len(indice_tuplas)

    print(f"Comienzo de entrenamiento con {epocas}
epocas.")
    for epoca in range(epocas):
        for i, (i_central, i_contextos) in enumerate(
indice_tuplas):
```

2b.1.9 Skip-gram: Implementación completa en Python

```
# ---Propagacion---
h = W1[i_central].reshape(-1, 1)
u = W2.T @ h
y = softmax_cp(u)

# ---Retropropagacion---
EI = y.copy()
EI[i_contextos] -= 1
W2 -= eta * (h @ EI.T)

EH = W2 @ EI
W1[i_central] -= eta * EH.T[0]

if i % 1000 == 0:
    print(f"Epoca {epoca}, Par: {i}/{total_pares}")

print(f"Fin de epoca: {epoca}")
```


2b.1.10 Skip-gram: Implementación completa en Python

```
# ---Guardado de Pesos---
if epoca % intervalo_guardado == 0 or epoca == epocas
- 1:
    nombre_archivo = f'pesos_skipgram_{nombre_pc}_epoca{
epoca}.npz'
    guardar_modelo(nombre_archivo, W1, W2, eta=eta, N=N,
C=C, cpararray=True)

print(f"Entrenamiento con {epocas} terminado.")
return W1, W2
```

2b.1.11 Skip-gram: Propagación hacia adelante

Fórmulas

$$h = W^T x = v_{p_I}, \quad u = W'^T h, \quad y = \text{softmax}(u)$$

Implementación en Python

```
# Propagacion
h = W1[i_central].reshape(-1, 1)
u = W2.T @ h
y = softmax_cp(u)
```

2b.1.12 Skip-gram: Cálculo del error

Pérdida

$$EI = y - t$$

donde t es el vector one-hot con 1 en la posición de la palabra de contexto.

Implementación en Python

```
# Retropropagacion
EI = y.copy()
EI[i_contextos] -= 1    # resta 1 en las posiciones del
contexto
```

2b.1.13 Skip-gram: Actualización de W'

Regla de actualización

$$v'_j \leftarrow v'_j - \eta E_{Ij} h$$

Implementación en Python

```
# Actualizacion de la matriz de salida W2  
W2 -= eta * (h @ EI.T)
```

2b.1.14 Skip-gram: Actualización de W

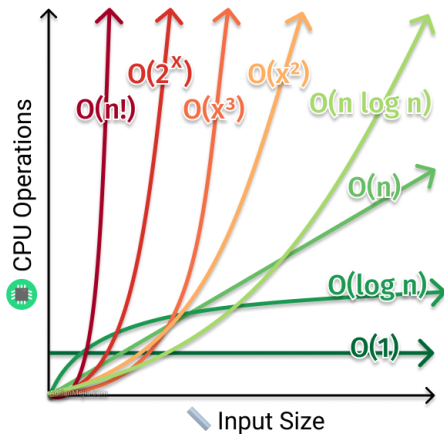
Regla de actualización

$$v_{p_I} \leftarrow v_{p_I} - \eta EH^T, \quad \text{donde } EH = W'EI$$

Implementación en Python

```
EH = W2 @ EI
W1[i_central] -= eta * EH.T[0]
```

Skip-gram (Negative Sampling)



2b.2.1 Skip-gram(NS)

Definición: Skip-gram con muestreo de ejemplos negativos (Negative Sampling)

Propósito: Reducir el costo computacional al actualizar los pesos en el modelo SkipGram.

Principio: Actualizar a W' aplicando la técnica de muestro de ejemplos negativos mencionada anteriormente

2b.2.2 Lógica de algoritmo

Función: `entrenar_skipgram_neg_samp()`

Al ejecutarse:

- Se carga el corpus y vocabulario
- Se inicializan los pesos W y W' .
- Se generan las tuplas (central, positivos, negativos).

encontraría a
negativos

la maga
positivos

?
central

tantas veces
positivos

me había
negativos

- Se **entrenan** los pesos por q épocas y por cada tupla generada.
- se guardan los pesos cada p épocas.

2b.2.3 Código de bucle de entrenamiento

```

for epoca in range(epocas):
    for i, (i_central, i_positivos, i_negativos) in enumerate(
        indice_tuplas):
        i_total = i_positivos + i_negativos

        h = W1[i_central].reshape(-1, 1)
        u = W2[:, i_total].T @ h
        y = sigmoide_np(u)

        EI = y.copy()
        EI[:len(i_positivos)] -= 1
        W2[:, i_total] -= eta * (h @ EI.T)

        EH = W2[:, i_total] @ EI
        W1[i_central] -= eta * EH.T[0]

```

2b.2.5 Dificultades

La función `sigmoide_np` generaba valores **NaN** en las matrices de pesos W y W' al actualizarlos.

```
def sigmoide_np(x):  
    return 1 / (1 + np.exp(-x))
```

Para solucionar esto utilizamos un valor de η menor (0,001).

2b.2.6 Funciones para resultados

Función para similitud del coseno: Cercanos a 1 indican que los vectores apuntan en una dirección muy similar, cercanos a 0 indican poca o ninguna relación y cercanos a -1 indican que los vectores apuntan en direcciones opuestas en el espacio vectorial, pero esto no implica necesariamente una relación semántica opuesta, sino solo una oposición geométrica.

```
def calcular_similitud(diccionario_palabras, W1, palabra1,
    palabra2):
    if palabra1 not in diccionario_palabras or palabra2 not in
        diccionario_palabras:
    return None

    indice1 = diccionario_palabras[palabra1]
    indice2 = diccionario_palabras[palabra2]
    embedding1 = W1[indice1]
    embedding2 = W1[indice2]
    similitud = embedding1 @ embedding2 / (linalg.norm(
        embedding1) * linalg.norm(embedding2))
    return similitud
```

2b.2.6 Funciones para resultados

Función para generar oraciones: Se propaga hasta softmax() con la idea de ir creando una oración según lo obtenido, se intenta evitar palabras repetidas, ya que se repetían constantemente.

```
def generar_oracion(corpus, W1, W2, diccionario_palabras,
    diccionario_indices, palabras_previas, longitud=10):
    palabras_previas_lista = palabras_previas.split()

    for _ in range(longitud):
        indices = [diccionario_palabras[p] for p in
            palabras_previas_lista if p in diccionario_palabras]
        if not indices:
            print("No se encontraron palabras previas en el
                diccionario.")
            break
        h = np.mean(W1[indices], axis=0).reshape(-1, 1)
        u = W2.T @ h
        y = softmax(u).flatten()
```

2b.2.6 Funciones para resultados

```
candidatos = np.argsort(-y)

palabra_siguiente = None
for idx in candidatos:
    candidata = diccionario_indices[idx]
    if candidata not in palabras_previas_lista:
        palabra_siguiente = candidata
        break

if palabra_siguiente is None:
    palabra_siguiente = diccionario_indices[candidatos[0]]

palabras_previas_lista.append(palabra_siguiente)

return " ".join(palabras_previas_lista)
return similitud
```

2b.2.7 Modelo entrenado y resultados

Modelo entrenado: Se utilizó un modelo tipo *skip-gram* con los siguientes hiperparámetros:

- Dimensión de embeddings: 65
- Ventana de contexto: 3
- Tasa de aprendizaje: 0.001
- Épocas: 950
- Negativos: 5

Similitud del coseno entre pares de palabras:

| Palabra 1 | Palabra 2 | Similitud |
|-----------|-----------|-----------|
| rey | reina | 0.6736 |
| gato | perro | 0.8626 |
| vida | muerte | 0.8583 |
| arriba | abajo | 0.9153 |
| feliz | triste | 0.8680 |
| coche | automóvil | 0.7134 |

2b.2.8 Generación de oraciones

A partir de distintas palabras iniciales, el modelo genera continuaciones basadas en los embeddings aprendidos:

Ejemplos:

- Palabras iniciales: *“el gato”*
Oración generada: *“el gato dónde codo arrestado lucas suponga maga argentada detenga agitación entrarían”*
- Palabras iniciales: *“la casa”*
Oración generada: *“la casa traveler arrestado suponga sin argentada detenga agitación entrarían tejado sobrepasara”*
- Palabras iniciales: *“un perro”*
Oración generada: *“un perro talita arrestado suponga maga argentada detenga agitación entrarían tejado sobrepasara”*