

# Trabajo Práctico 2

## Aprendizaje Automático Avanzado

Cisnero Matias, Seivane Nicolás, Serafini Franco  
20 de Octubre de 2025

## Modificación de Corpus



# Idea general

El objetivo es mejorar la representación del corpus uniendo palabras que aparecen juntas con alta frecuencia en el mismo contexto.

- Buscar palabras frecuentes.
- Analizar sus contextos más comunes.
- Unirlas si aparecen juntas frecuentemente.

# Funciones principales

```
def palabras_frecuentes_en_contexto(corpus, palabra_objetivo
    , contexto=1):
    frecuencias = {}
    for i in range(len(corpus)):
        if corpus[i] == palabra_objetivo:
            for j in range(i - contexto, i + contexto + 1):
                if j != i and 0 <= j < len(corpus):
                    palabra_contexto = corpus[j]
                    frecuencias[palabra_contexto] = frecuencias.get(
palabra_contexto, 0) + 1
    return sorted(frecuencias.items(), key=lambda x: x[1],
        reverse=True)
```

# Visualización del contexto



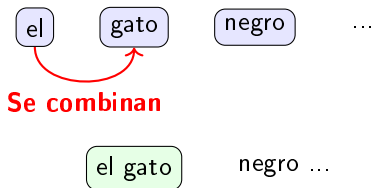
Contexto = 1 alrededor de negro

# Unir palabras frecuentes en contexto

```
def unir_palabras_en_contexto(corpus, palabra1, palabra2):
    nuevo_corpus = []
    i = 0
    while i < len(corpus):
        if corpus[i] == palabra1 and i + 1 < len(corpus) and
        corpus[i + 1] == palabra2:
            nuevo_corpus.append(f"{palabra1} {palabra2}")
            i += 2
        else:
            nuevo_corpus.append(corpus[i])
            i += 1
    return nuevo_corpus
```

## Ejemplo visual de unión

- Si ambas palabras aparecen frecuentemente juntas y una de ellas suele ser predicha mal muchas veces se hace lo siguiente, por ejemplo: Si el **token** 'el' es frecuentemente mal predicho, entonces se une a una palabra que frecuente mucho como 'gato'. No necesariamente se hace de derecha a izquierda.



## Bucle principal de optimización

```
def optimizar_corpus(words, min_frecuencia=200, contexto=1,
    top_contextos=3, iteraciones=3, frecuencia_min=100):
    corpus_modificado = words.copy()
    for iteracion in range(iteraciones):
        cuenta = contar_palabras(corpus_modificado)
        palabras_objetivo = cuenta[cuenta > min_frecuencia].
        index
            for palabra_objetivo in palabras_objetivo:
                resultados = palabras_frecuentes_en_contexto(
                    corpus_modificado, palabra_objetivo, contexto)
```



# Bucle principal de optimización

```
for palabra_contexto, frecuencia in resultados[:  
    top_contextos]:  
    if frecuencia > frecuencia_min:  
        corpus_modificado = unir_palabras_en_contexto(  
            corpus_modificado, palabra_objetivo, palabra_contexto)  
return corpus_modificado
```

## Caso 1: Palabra no suficientemente frecuente

- La palabra no alcanza la frecuencia mínima `min_frecuencia` para ser considerada.
- No se analizan sus contextos, ni se combinan tokens.

cucharón

de

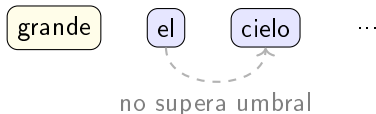
sopa

...

No se combina: frecuencia insuficiente

## Caso 2: Palabra entra al bucle, pero sus contextos no califican

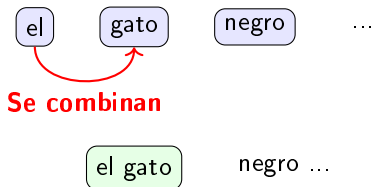
- La palabra tiene frecuencia suficiente para entrar al análisis.
- Sin embargo, las palabras de su contexto no superan `frecuencia_min`, por lo que no se unen.



No se unen pese a aparecer juntas

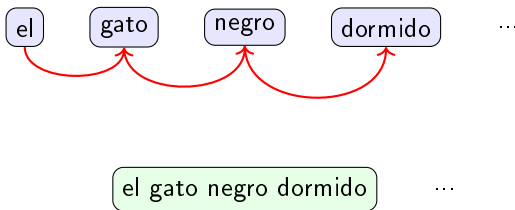
## Caso 3: Palabras frecuentes se combinan

- Ambas palabras son frecuentes y aparecen juntas.
- Se cumple el umbral de contexto, por lo que se unen.



## Caso 4: Uniones sucesivas en cadena

- Varias palabras frecuentes cumplen las condiciones y se van uniendo sucesivamente.
- Se produce un efecto en cadena en el corpus optimizado.



# Fragmentación

T o m a t o e s a r e o n e o f t h e m o s t p o p u l a r p l a n t s f o r v e g e t a b l e g a r d e n s .  
 T i p f o r s u c c e s s . I f y o u s e l e c t v a r i e t i e s t h a t a r e r e s i s t a n t t o d i s e a s e a n d  
 p e s t s , g r o w i n g t o m a t o e s c a n b e q u i t e e a s y . F o r e x p e r i e n c e d g a r d e n e r s  
 l o o k i n g f o r a c h a l l e n g e , t h e r e a r e e n d l e s s h e i r l o o m a n d s p e c i a l t y  
 v a r i e t i e s t o c u l t i v a t e . T o m a t o p l a n t s c o m e i n a r a n g e o f s i z e s .

Figura: Imagen extraída de IBM watsonx

## Caso 1: Ningún token coincide con el vocabulario

- Se recorren las palabras del texto pero ningún token está en el vocabulario.
- La función devuelve `None` tras imprimir el aviso.

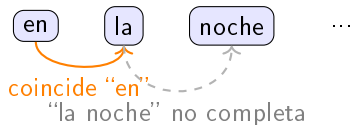
brillan      las      estrelinas      ...

↓ no está en vocab  
↓

Retorna `None` — palabra fuera del vocabulario

## Caso 2: Coincidencias parciales pero no completas

- Se encuentra una parte del texto en el vocabulario, pero no el token más largo posible.
- El bucle continúa buscando sin romper correctamente en la posición óptima.

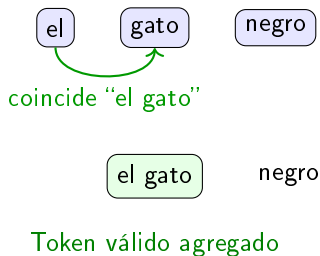


Solo se fragmenta lo que se encuentra



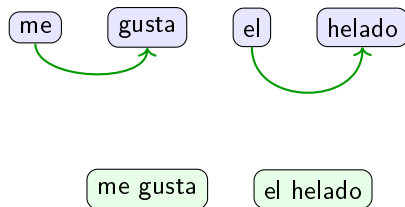
## Caso 3: Tokenización exacta con vocabulario

- Se encuentra una secuencia exacta en el vocabulario.
- El token se agrega a la lista de salida correctamente.



## Caso 4: Coincidencias largas y fragmentación en secuencia

- Se van encontrando coincidencias de mayor cantidad de tokens unidos (priorizando tokens más largos).
- Se produce una fragmentación por grupos que maximiza coincidencias.



Tokenización óptima por grupos

# Fragmentación

```
def tokenizar_por_vocab(texto, vocab, indices = False):
    palabras = texto.lower()
    palabras = re.findall(r'\w+|[\^\\w\\s]', palabras, flags=re.
        UNICODE)
    tokens = []
    i = 0
    n = len(palabras)

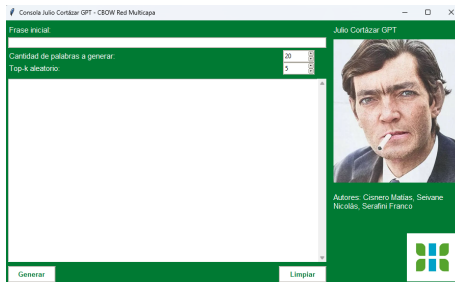
    while i < n:
        cand_final = None
        for j in range(n, i, -1):
            cand = " ".join(palabras[i:j])
            if cand in vocab:
                cand_final = cand
                i = j
                break
```

# Fragmentación

```
if not cand_final:
    cand_final = palabras[i]
    if cand_final not in vocab:
        print(f'palabra: [{cand_final}] no esta en voabulario')
        return None
    i += 1

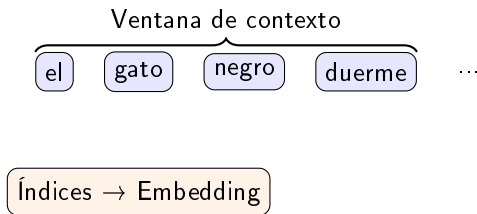
if indices is False:
    tokens.append(cand_final)
else:
    tokens.append(palabras_a_indice[cand_final])
return tokens
```

# Generación de texto



# CBOW (One-Hot): Construcción de la ventana de contexto

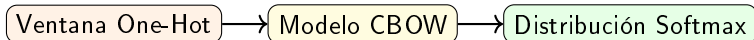
- Las últimas 10 palabras del texto se convierten en índices de vocabulario.
- Si hay menos de 10, se repite la última palabra para completar la ventana.



Cada palabra → vector de embedding

## CBOW (One-Hot): Predicción de palabra siguiente

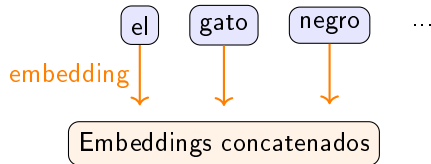
- El modelo predice un vector de embedding para la palabra objetivo.
- Devuelve una distribución de probabilidad (softmax), del tamaño one-hot del vocabulario.
- Se elige aleatoriamente una palabra del top- $k$ .



Se elige la palabra más probable (o aleatoria del top- $k$ )

## CBOW (Embeddings): Construcción de la ventana

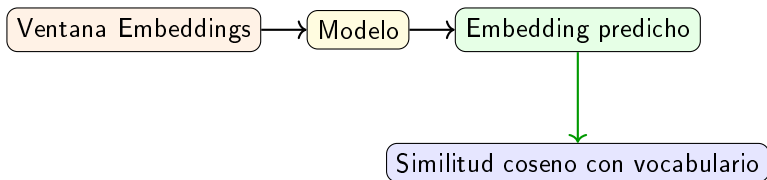
- Como en el caso anterior, se concatenan los **embeddings** de cada palabra.
- Si hay menos de 10 palabras, se repite la última embedding.





## CBOW (Embeddings): Predicción y similitud

- El modelo predice un vector de embedding para la palabra objetivo.
- Se calcula la similitud coseno con todos los embeddings del vocabulario.
- Se elige la palabra más similar (o una del top- $k$ ).



Se elige la palabra más similar

# Predicción One-hot

```
def predecir_cbow_onehot(palabras, modelo, indice_a_palabras
    , indices_a_embeddings, palabras_a_indice, topk=5):

    palabras_a_indice = globals().get('palabras_a_indice')

    tokens_idx = tokenizar_por_vocab(palabras,
        palabras_a_indice, indices=True)

    if tokens_idx is None or len(tokens_idx) == 0:
        return None

    if len(tokens_idx) < 10:
        tokens_idx = tokens_idx + [tokens_idx[-1]] * (10 - len(
            tokens_idx))
    else:
        tokens_idx = tokens_idx[-10:]
```

# Predicción One-hot

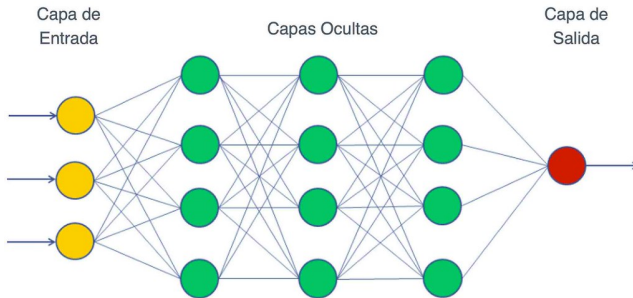
```
ventana = np.concatenate([indices_a_embeddings[idx] for idx
    in tokens_idx]).flatten()
pred = modelo.predict(ventana.reshape(1, -1), verbose=0)
probs = np.asarray(pred).flatten()

candidatos = np.argsort(-probs)
topk_indices = candidatos[:topk]
top1 = np.random.choice(topk_indices)
palabra = indice_a_palabras[top1]
return palabra
```

# Predicción Representación Contextual

```
ventana = np.concatenate([W[idx] for idx in tokens_idx]).  
    flatten()  
pred_emb = modelo.predict(ventana.reshape(1, -1), verbose=0)  
pred_emb = np.asarray(pred_emb).flatten()  
  
sims = cosine_similarity(pred_emb.reshape(1, -1), W)[0]  
  
topk_idx = np.argsort(-sims)[:topk]  
  
top1 = np.random.choice(topk_idx)  
palabra_predicha = indice_a_palabras[top1]  
  
return palabra_predicha
```

# Estructura Multicapa



Créditos: Antonio Richaud

# Estructura Multicapa

**Modelo:** se implementó una red neuronal de tipo **Perceptrón Multicapa (PMC)** utilizando la librería **Keras** (TensorFlow).

**Entradas:** vector resultante de concatenar los embeddings de una **ventana de 8 palabras previas**, donde cada palabra está representada por su embedding individual.

**Salida:** vector correspondiente a la palabra objetivo del contexto.

**Arquitectura:**

- Capa oculta 1: 512 neuronas, activación gelu.
- Capa oculta 2: 256 neuronas, activación gelu.
- Capa oculta 3: 128 neuronas, activación gelu.
- Capa de salida:  $N$  neuronas, activación sigmoid.

**Optimizador:** Adam con learning rate = 0.0001.

**Función de pérdida:** MSE (error cuadrático medio).

# Entrenamiento del Modelo

El modelo fue entrenado con el corpus embebido, aplicando la función sigmoide a los datos para su normalización.

## Parámetros principales:

- Épocas: 250
- Ventana de contexto: 8 palabras
- Error mínimo alcanzado: **0.0794**

# Generación de Texto

La generación de texto se realiza a partir de una secuencia inicial de palabras.

## Código:

```
poner codigo
```

La función realiza:

- 1 Lectura de la secuencia inicial.
- 2 Predicción de la siguiente palabra según el tipo de salida.
- 3 Evita repeticiones consecutivas de palabras.
- 4 Actualiza la ventana de contexto y repite hasta completar la longitud deseada.



# Resultados de la Predicción

Dado un conjunto de 8 palabras previas, el modelo entrenado con Keras fue capaz de generar una secuencia de palabras consecutivas dentro del corpus.

## Entrada inicial:

*“hola como esta usted la noche de hoy”*

## Texto generado:

*“diarios mirará saliéramos encegueció creas autopista . admirativamente , saliéramos encegueció feudal creas saliéramos admirativamente encegueció bebida saliéramos contaba encegueció creas saliéramos honorable encegueció creas saliéramos honorable encegueció creas saliéramos”*