UNIVERSIDAD NACIONAL
DEL NORDESTE

CONICET

I M I T

Clase 5:

Procesamiento de lenguaje natural (NLP)

Grandes modelos de lenguaje (LLMs)

Objetivos

- Procesamiento de lenguaje
- Atención
- Transformers
- Primera ola de LLMs: BERT,T5
- Segunda ola de LLMs: GPT, LLAMA, MIXTRAL, CLAUDE

Bibliografía

- Jurafsky and Martin, 2023: Speech and Language Processing. (Draft/Open)
- Zhang, A., Lipton, Z.C., Li, M. and Smola, A.J., 2021. Dive into deep learning. arXiv preprint arXiv:2106.11342.
- Prince, 2023. Understanding deep learning.

Datos para NLP

Las bases de datos de NLP estan compuestas por textos, que pueden ser libros, artículos, paginas web, sentencias judiciales, etc, o todos estos juntos.
La base de datos se denomina el "corpus" o si considero varias, "corpora".
Todas las palabras que hay en un corpus conforman el vocabulario.
¿Cual es el dato individual con el que vamos a trabajar? Como subdividimos al corpus?
Carácteres?, Palabras?, Grupo de palabras?, Oraciones?

Librerías para el preprocesamiento: NLTK, Open NLP, Gensim, scikit-learn.

Preprocesamiento

Queremos un texto formateado para que pueda ser entendible por la máquina.

- Limpieza general. Cuestiones de formato, negritas, símbolos extraños.
- Separación en oraciones. El punto seguido marca una división de contextos. El punto aparte una mayor aun.
- Stemming or lemmatization. Raiz de las palabras, verbos en infinitivo, etc
- Stop-word removal. Artículos, preprosiciones, etc son comunes y no hacen al fondo de la cuestión.
- Corrección de errores ortográficos. Acentos?. Proceso esencial en mensajes, whatsapps, emails, reclamos, etc.
- Tokenization. Dependen del idioma...

Tokens y tokenización

- Trabajar con un diccionarios de palabras:
    - Hay muchas palabras derivadas del mismos significado (Ej conjugacion de verbos) .
    - Hacen falta los signos de puntuación.
- Trabajar a nivel de carácteres:
    - Se construyen palabras muy facilmente.
    - Agregamos un nivel de complejidad muy grande.

El método utilizado es algo intermedio, los tokens:

- Se tienen palabras simples sueltas.
- Se tienen raíces de palabras combinadas.
- Se tienen los signos de puntuación.

Entonces token es la unidad basica de nuestro modelo de lenguaje.

Tokenizador: dado un corpus tiene que identificar los tokens. Generalmente se pone un numero maximo de tokens a priori. Se van construyendo tokens a traves de armado de las palabras con los caractéres.

Ej. Word-piece.

Marco probabilístico para lenguaje

"El señorial chalet se encontraba a la izquierda de la. . . "

Supongamos queremos ver si la proxima palabra es adecuada de acuerdo a la sentencia que vengo escribiendo:

$$p(w_{1:n}) = p(w_1)p(w_2|w_1)\ldots p(w_n|w_{1:n-1}) = \prod_{k=1}^{n} p(w_k|w_{k-1})$$

‘ Notación para sequencias: $p(w_{1:n}) \doteq p(w_1, \cdots, w_n)$

Vamos construyendo la sentencia (sampleando) en base a las palabras que ya tenemos escritas.

Tenemos a Cervantes!

La primera la sacamos de la galera?

Si tengo que responder preguntas o hablar sobre algo particular?

N-gramas

El lenguaje es demasiado creativo. ¿Cuantas veces en el corpus voy a tener
"El señorial chalet se encontraba a la izquierda de la escuela"?
Para sentencias muy largas, las primeras palabras de la sentencia no deberían influir. Me olvido de la historia vieja y solo recuerdo las últimas palabras:

$$p(w_{1:n}) = p(w_1)p(w_2|w_1)p(w_3|w_1,w_2)p(w_4|w_2,w_3)\cdots p(w_n|w_{n-1},w_{n-2})$$

La aproximación que estoy haciendo es: $p(w_n|w_{1:n-1}) \approx p(w_n|w_{n-2:n-1})$.
En este caso solo estoy considerando los dos tokens pasados para medir la probabilidad del corriente.
Esto es lo que se conoce como un trigrama, la probabilidad de la palabra corriente condicionada a los dos últimos tokens.

N-gramas

Usando regla de Bayes la probabilidad $p(w_n|w_{n-1}, w_{n-2})$ puede ser interpretada como:

$$p(w_n|w_{n-1}, w_{n-2}) = \frac{p(w_n, w_{n-1}, w_{n-2})}{p(w_{n-1}, w_{n-2})}$$

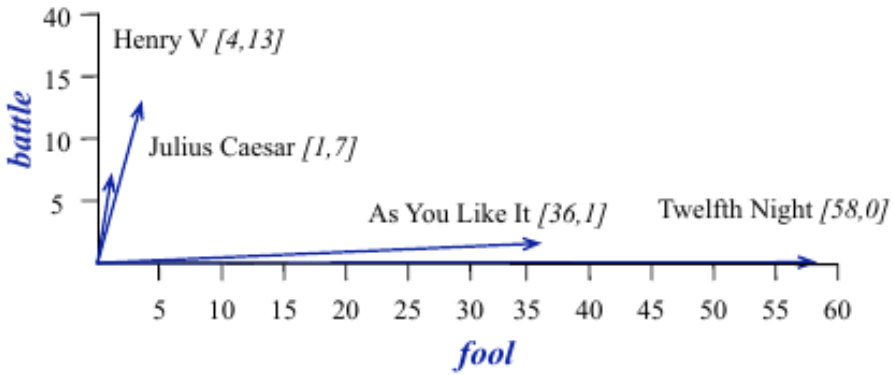
Contando cuantos de estos bigramas y trigramas hay en el corpus puedo calcular la probabilidad $p(w_n|w_{n-1}, w_{n-2})$:

$$p(w_n, w_{n-1}, w_{n-2}) = \frac{\text{Nro de veces de} \{w_{n-2}, w_{n-1}, w_n\}}{\text{Nro total de trigramas}}$$

Matriz de coocurrencia

Matriz de coocurrencia documento-palabra.

- Pongo todas las palabras en las filas y los documentos en las columnas y hago conteo.
- Cada fila puede ser un vector de representacioń de la palabra



Matriz de coocurrencia palabra-palabra.

El elemento cuenta cuantas veces coocurren en el contexto (4-10 palabras alrededor de la palabra).

Matriz de coocurrencia

Idea: palabras que aparecen juntas con frecuencia en un contexto tienen una relación semántica mas cercana que las palabras que rara vez aparecen juntas.

Se puede utilizar para calcular medidas de similitud entre palabras.

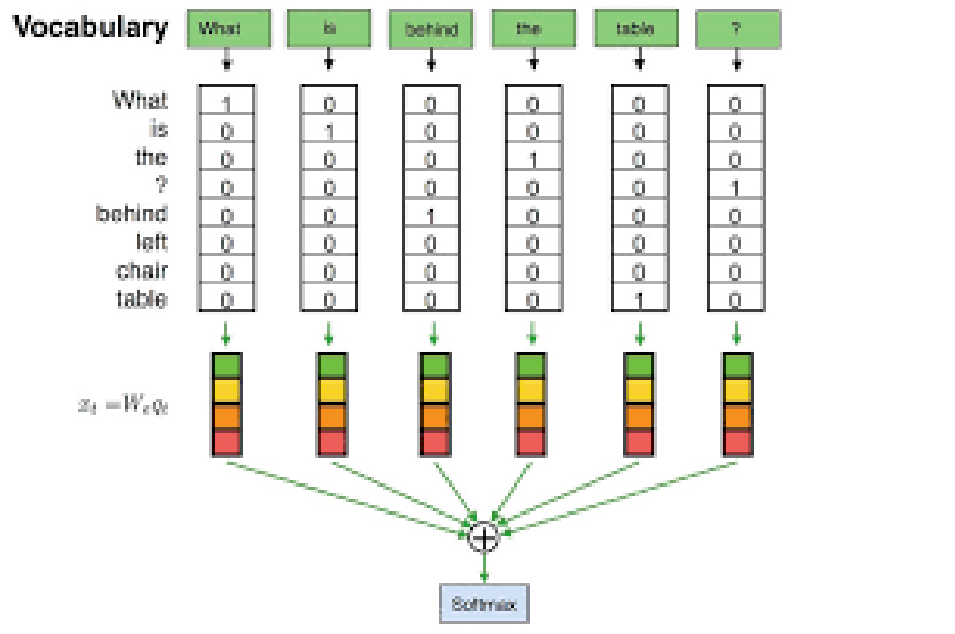
Se puede utilizar para visualizar el espacio semántico de las palabras (figura).

¿Como se logró graficar las palabras en 2D?

Bag of words - Sopa/Bolsa de palabras

- Otra forma de representar las palabras en vectores es un espacio del tamaño del vocabulario.
- Cada palabra en este espacio sería un 0 y en el lugar de la palabra un 1.
- Las oraciones se representan sumando los vectores de cada palabra.
- No hay sentido de tiempo/secuencia

	about	bird	heard	is	the	word	you
About the bird, the bird, bird bird bird	1	5	0	0	2	0	0
You heard about the bird	1	1	1	0	1	0	1
The bird is the word	0	1	0	1	2	1	0



Esto es el denominado Bag Of Words/BOW

Medida de similaridad

Si asumimos dos tokens se encuentran representados en un espacio vectorial, como definimos cuan similares son estos tokens?

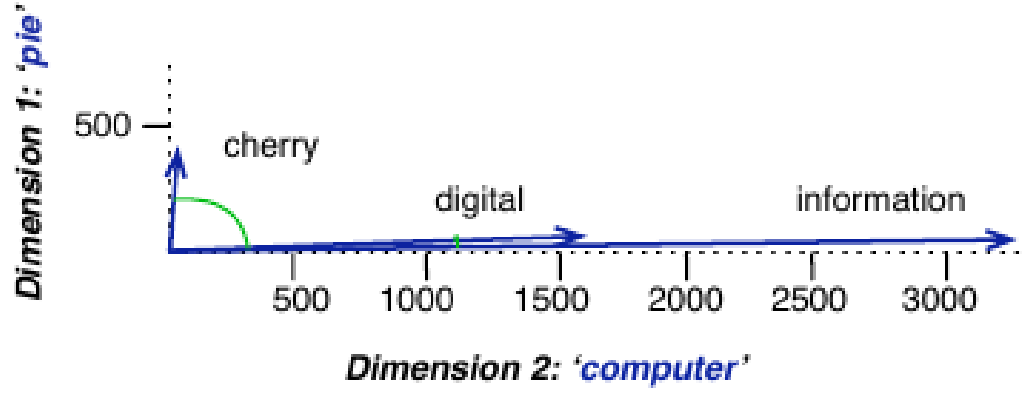
La medida de similaridad mas utilizada es cuanto se superponen estos vectores, esto puede medirse con el producto punto:

$$\mathbf{v} \cdot \mathbf{w} = \sum_{i}^{N} v_i w_i$$

Si queremos hacerlo independiente de la magnitud normalizamos con las longitudes de los vectores,

$$\cos(\mathbf{v} \cdot \mathbf{w}) = \frac{\mathbf{v} \cdot \mathbf{w}}{|\mathbf{v}||\mathbf{w}|}$$

	pie	data	computer
cherry	442	8	2
digital	5	1683	1670
information	5	3982	3325



TF-IDF. Term Frequency - Inverse Document Frequency

La matriz de coocurrencia token-document mide solo un conteo, pero no considera cuanta información agrega la palabra. Palabras frecuentes no siempre agregan significado.

Contamos las palabras en los documentos como antes:

$$TF_{t,d} = \log_{10}(1 + \text{count}(t, d))$$

El $\log$ es para achatar conteos grandes.

Queremos dar mayor peso a las palabras que ocurren en unos pocos documentos:

$$IDF_{t,d} = \log_{10}\left(\frac{N_d}{\text{count}(d, t)}\right)$$

N_d número total de documentos, $\text{count}(d, t)$ cantidad de documentos donde aparece t.

Si la palabra t aparece en todos los documentos $\text{count}(d, t) = N_d$ luego $IDF_{t,d} = 0$!!!

Embeddings

La representación de la matriz de coocurrencia tiene un enorme problema, la dimensión es equivalente a la longitud del vocabulario. El espacio es esparso.
Ejemplo: *"El señorial chalet se encontraba a la izquierda de la. . . "*
Existe un conjunto de palabras mas probables "casa, escuela, zapateria, etc".

- Palabra con funciones/significados/atributos similares deberían encontrarse "cerca" en el espacio.
- Embeddings son representaciones de las palabras en espacios de mas baja dimensionalidad y densos

Skip-gram. Word2vec

Entrenamos un clasificador binario para predecir cual es la probabilidad de que la palabra w aparezca cerca de una candidato c.
Usando el producto interno como similaridad, la probabilidad de que c este en el contexto de w es:

$$p(+|w,c) = \frac{1}{1+\exp(-\mathbf{c}\cdot\mathbf{w})}$$

Los pesos de esta red son la representación de la palabra. Este es el embedding.
Es sencillo clasificar porque usamos las próximas palabras como targets.

Word2vec

- Los inscrutamientos se pre-entrenan.
- Skip-gram/word2vec son inscrustamientos estáticos.
- Aprende un feature/vector para cada palabra del vocabulario.

Método:

1. Trata la palabra target y el contexto como ejemplos positivos.
2. Se generan muestras aleatorias con palabras para generar muestras negativas.
3. Se usa la regresión logística para entrenar el clasificador para distinguir las muestras.
4. Se usan los pesos aprendidos como los embeddings.

RNNs for NLPs

El problema de predicción de la próxima palabra,

"El señorial chalet se encontraba a la izquierda de la. . . "

puede ser traducido a una probabilidad secuencial: $p(\mathbf{w}_k|\mathbf{w}_{1:k-1})$

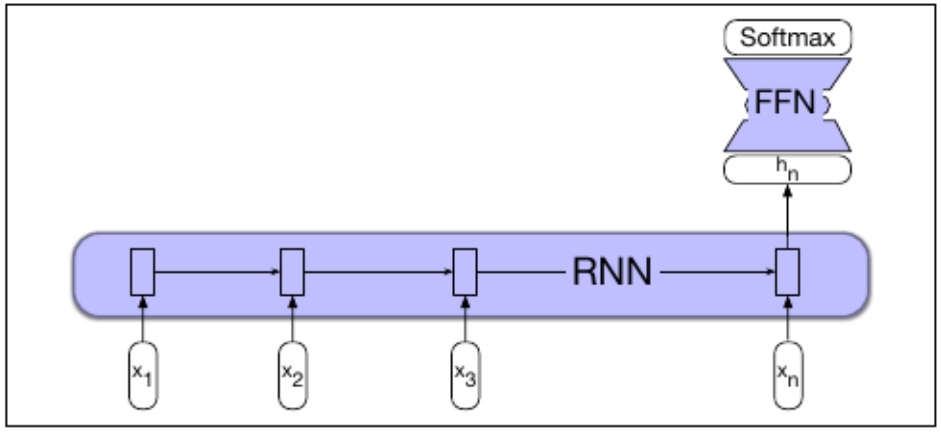
Predecimos la próxima palabra dadas las anteriores palabras de la oración $\rightarrow$ una RNN

El entrenamiento de la RNN:

- Separamos en sentencias el corpus.
- Hacemos entrenamiento autosupervisado con la próxima palabra de la sentencia. Conocemos el target.
- Las entradas a la red para cada tiempo son los embeddings pre-entrenados de cada palabra/token.
- Usamos cross entropy de función de pérdida:

$$J = -\sum_{w} y_k[w] log \hat{y}_k[w]$$

Análisis de sentimiento con una RNN



Queremos clasificar un texto/sentencia para analisis de sentimiento:

Para saber si es una opinión positiva/negativa.

- Ponemos a predecir la RNN la última palabra de la sentencia.
- Usamos el último estado latente de la RNN como input a una FCNN.
- Ponemos una softmax a la salida para dar la probabilidad positiva/negativa.

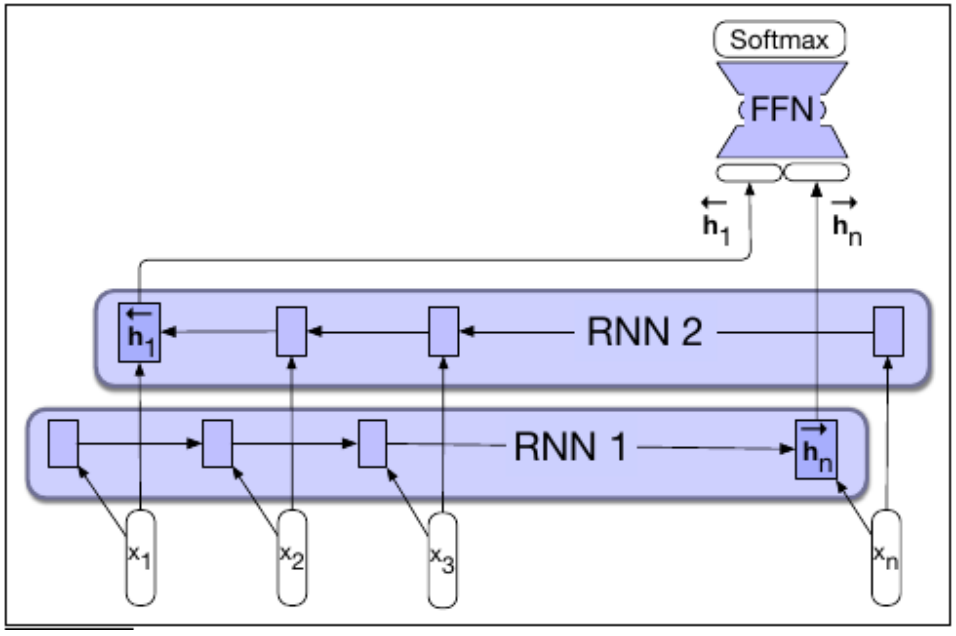
Bidireccional RNNs

El contexto no es solo lo ya dicho sino la última parte de la sentencia. El objetivo de la predicción es puramente encontrar un espacio latente que represente significados de acuerdo a contextos.

Podemos realizar una predicción de atrás hacia adelante: $p(x_k|x_{k+1:n})$

$$RNN_f(x_{1:k}) \rightarrow h_k^f$$

$$RNN_b(x_{k:n}) \rightarrow h_k^b$$



Tenemos dos espacios latentes uno con el contexto hacia adelante y otra hacia atrás.

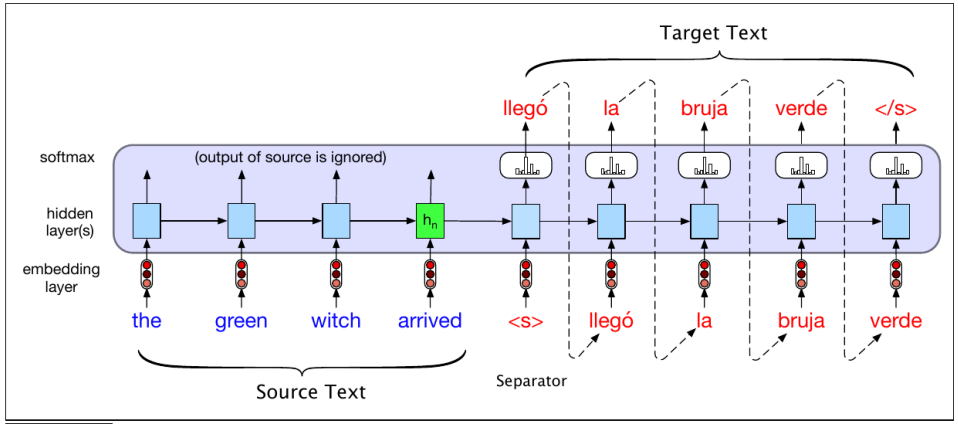
Finalmente concatenamos los dos estados/espacios latentes.

Encoder-decoder con RNNs

Cho et al. (2014), Sutskever et al. (2014) proponen usar concepto de autoencoder pero con RNNs (sequence to sequence).

Una RNN para ir al espacio latente otra para predecir oraciones de salida.

Usado con LSTMs para traducciones de Inglés a Francés. BLEU=34.8.

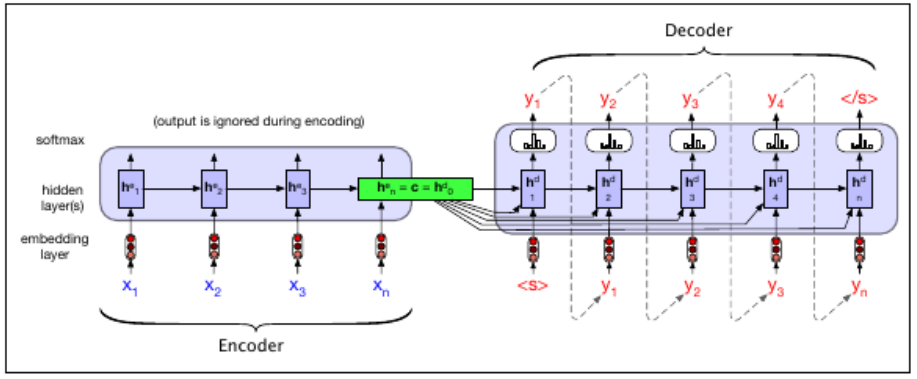


Dada una cadena de tokens de entrada, $\mathbf{x}_{1:n}$ queremos predecir n tokens de salida, $\mathbf{y}_{1:N}$.

Lo primero que hacemos es el embedding en el espacio, luego en base a las entradas generamos un vector de contexto.

El último estado latente del encoder h_n^e es pasado como estado latente inicial h_0^d al decoder.

Encoder-decoder con RNNs



Machine translation.

Entrenamiento con traducciones correctas (supervisado). Requiere datos manuales.

Muchas veces las palabras cambian el orden.

Cuando el texto de “traducción” es largo se puede perder el contexto.

Se puede pasar el h_0^d a toda la secuencia del decoder:

$$h_k^d = g(y_{k-1}, h_{k-1}^d, h_0^d)$$

$$z_k = f(h_k^d)$$

$$y_k = \text{softmax}(z_k)$$

La softmax sobre las posibles palabras del vocabulario por lo que nos da la probabilidad de la próxima palabra (usando el embbeding^{-1})

Significado de acuerdo al contexto

- Fui al banco para abrir una cuenta de ahorros.
- Nos sentamos en un banco del parque bajo la sombra de un timbo para descansar.
- Cruce todo el rio nadando para llegar al banco de arena que se encuentra del otro lado.

¿Cómo nos damos cuenta cuando leemos del signicado de banco?

Significado de acuerdo al contexto

- Fui al banco para abrir una cuenta de ahorros.
- Nos sentamos en un banco del parque bajo la sombra de un timbo para descansar.
- Cruce todo el río nadando para llegar al banco de arena que se encuentra del otro lado.

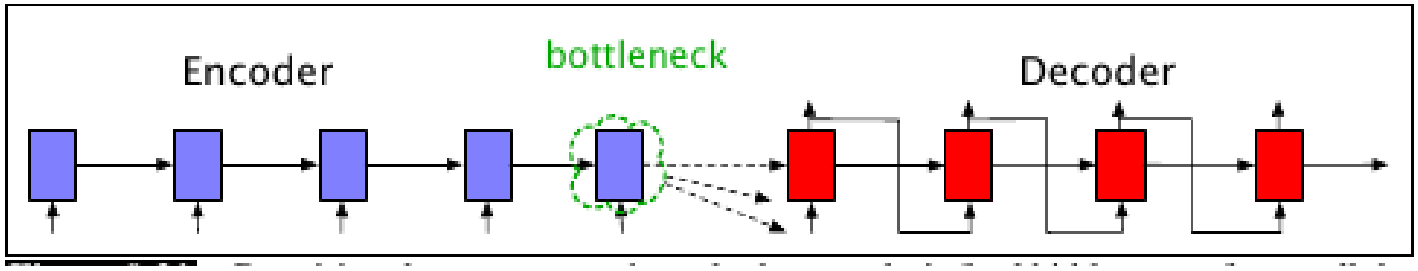
Hay un conjunto de “palabras claves” en el contexto que nos permiten determinar el significado de la palabra banco.

Dado que el significado varía sustancialmente de acuerdo al contexto, requiero de un embedding que tenga en cuenta el contexto.
$\Rightarrow$ La misma palabra en diferentes contextos tendrá asociado vectores distintos.

El embedding de word2vec (y cnia) es estático. No nos sirve para este propósito.

Problemas de las RNNs

El encoder-decoder con RNNs tiene una restricción, un fuerte peso de las últimas palabras, pero muchas veces en el lenguaje lo mas significativo puede estar en distintas posiciones.



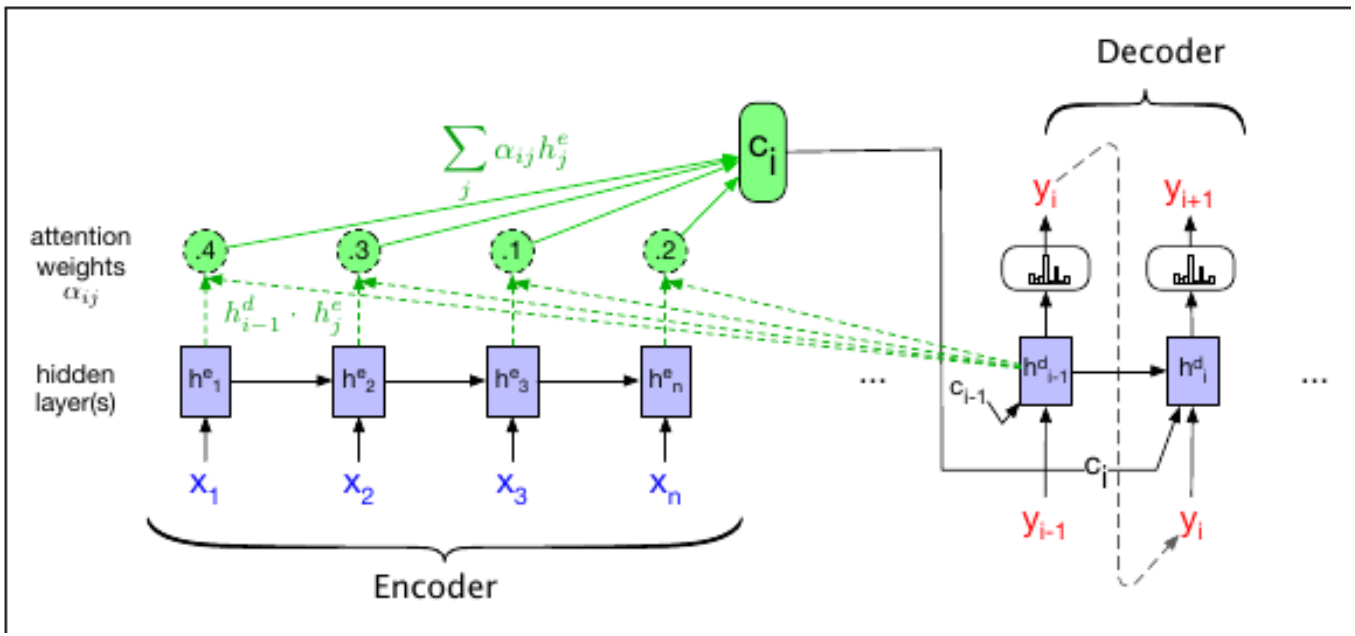
Consideremos a todas las entradas con pesos para armar la entrada del decoder.

Mecanismo de atención

Proponemos darles un peso a todas las palabras del contexto N_t:

$$\tilde{\mathbf{x}}_i = \sum_{j=1}^{N_t} \alpha_{ij}(\mathbf{x}_i)\mathbf{x}_j$$

El peso no es fijo depende del input.



- Los pesos permiten tener en cuenta si es mas significativo el contexto de la primera parte o de la segunda.
- El orden de las palabras es irrelevante.

Pesando el contexto de acuerdo a su relevancia

Supongamos que estamos tratando de inferir la salida i-ésima por lo que las entradas son $\mathbf{x}_{1:i}$, pesamos todas las entradas anteriores de acuerdo a su relevancia/similitud:

$$\begin{aligned}\alpha_{ij} &= \text{Softmax}(\mathbf{x}_i \cdot \mathbf{x}_j) \\ &= \frac{\exp\left(\mathbf{x}_i \cdot \mathbf{x}_j\right)}{\sum_{k=1}^{i} \exp\left(\mathbf{x}_i \cdot \mathbf{x}_k\right)}\end{aligned}$$

con $j \leq i$

Entonces la salida la pesamos de acuerdo a las entradas con una superposición lineal:

$$\mathbf{y}_i = \sum_{j=1}^{i} \alpha_{ij}\mathbf{x}_j, \qquad \mathbf{Y} = \boldsymbol{\alpha}\mathbf{X} = \text{Softmax}(\mathbf{X}\mathbf{X}^\top)\mathbf{X}$$

Recordar que estamos en un espacio donde el concepto de cercanía significa similitud de las palabras.

Query, keys y values

Supongamos que queremos confeccionar un recomendador de películas automático basado en las preferencias de los usuarios.

- key: Para cada película vamos a poner un conjunto de atributos que la clasifican: director, actores, genero, argumento.
- query: Ahora cada cliente tendra una query, que son los gustos del usuario, director, actores, etc.
- value: Los values son las peliculas en sí.

Entonces si las queries son cercanas a las keys estamos ante un “match”.

Query, keys y values

En lugar de pesar a las distintas “palabras”

$$\mathbf{Y} = \text{Softmax}(\mathbf{X}\mathbf{X}^\top)\mathbf{X}$$

Transformo los pesos y los values con estos conceptos:

$$\mathbf{Y} = \text{Softmax}(\mathbf{Q}\mathbf{K}^\top)\mathbf{V}$$

donde $\mathbf{Q} = \mathbf{X}\mathbf{W}^q$, $\mathbf{K} = \mathbf{X}\mathbf{W}^k$, $\mathbf{V} = \mathbf{X}\mathbf{W}^v$.

Los pesos $\mathbf{W}^q, \mathbf{W}^k, \mathbf{W}^v$ son redes neuronales que dependen del input.

Positional encoding

La posición en una oración es importante:

1. Vamos a ir al rio en lugar de ir a la facu.
2. Vamos a ir la facu en lugar de ir al rio.

Necesito decirle al modelo en que lugar se encuentra el token.

Definimos posición con $\mathbf{r}$.

En lugar de aumentar la dimensionalidad vamos a mezclar la información: $\tilde{\mathbf{x}} = \mathbf{x} + \mathbf{r}$

Lo escribimos con funciones sinusoidales supongamos la dimensión del embedding es D:

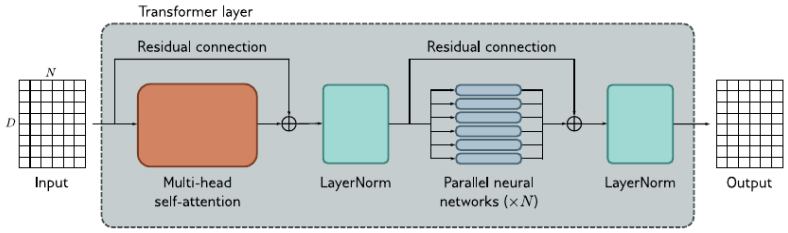
$$r_{ni} = \sin\left(\frac{n}{L^{i/D}}\right)$$

$$r_{ni} = \cos\left(\frac{n}{L^{(i-1)/D}}\right)$$

donde $i = 1, \cdots, D$.

Esto es el mismo concepto que escribir un punto de retícula con la transformada de Fourier.

Transformer

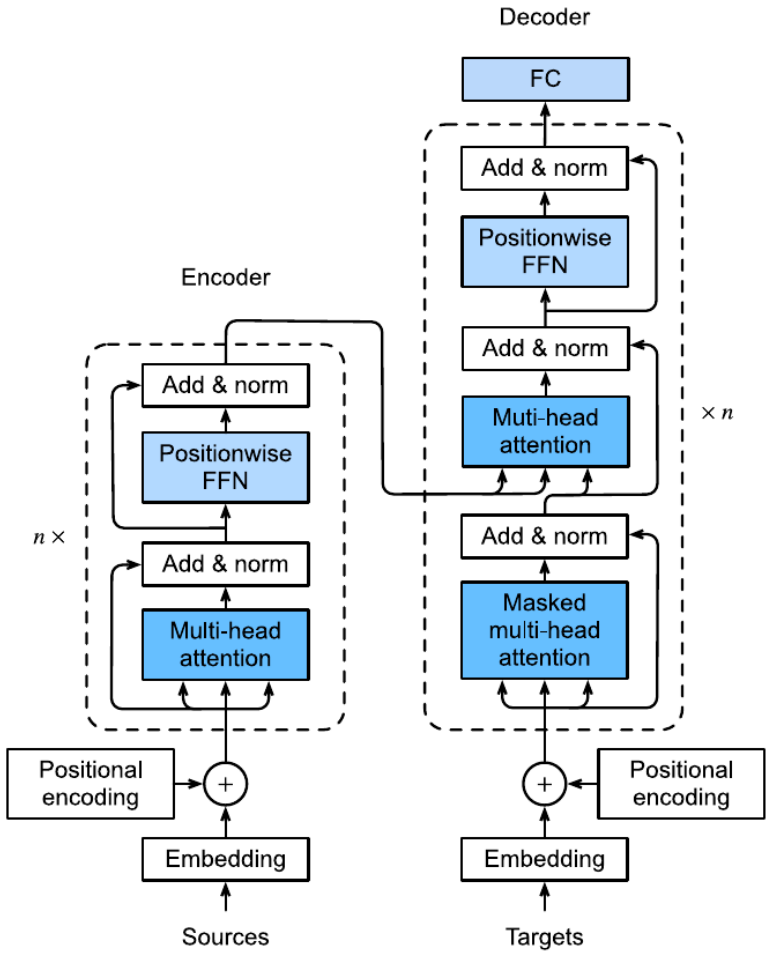


$$X = X + MhSa[X],$$

$$X = LayerNorm[X],$$

$$xn = xn + mlp[xn],$$

$$X = LayerNorm[X]$$

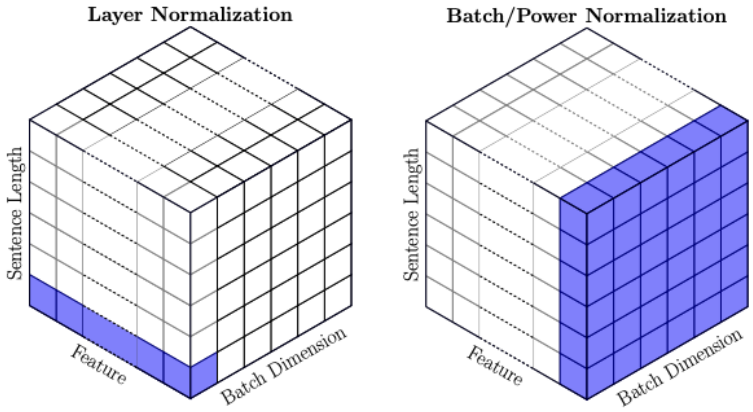


Estructura/Bloque basada en self-attention combinada con una red completamente conectada MLP/FCNN y normalización de capas.

Normalización de capas

En lugar de normalizar la entrada (normalizacion del batch), en NLP se normaliza la capa (los features).

En el caso del batch el largo es siempre el mismo pero en NLP tenemos variaciones de la longitud de la sentencia, cambiando la constante de normalización.

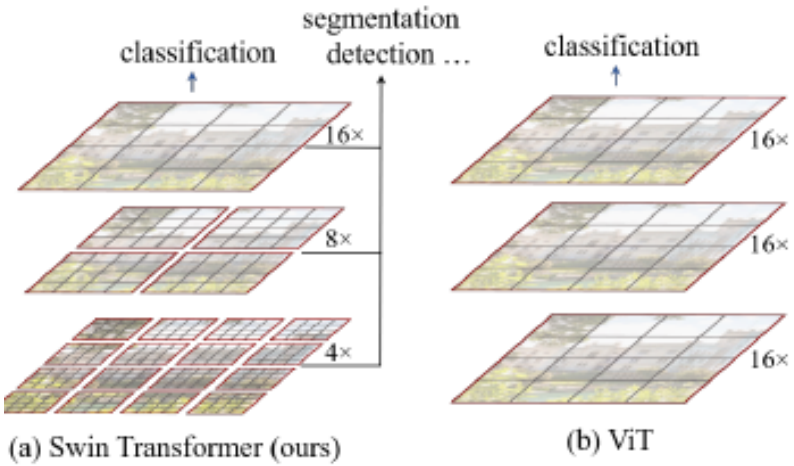


Swin Transformers

Los transformers se han difundido a otras aplicaciones mas alla de NLP. En visión no se puede aplicar self-attención directamente.

Transformers jerárquicos con Shifted WINdows(Swin).

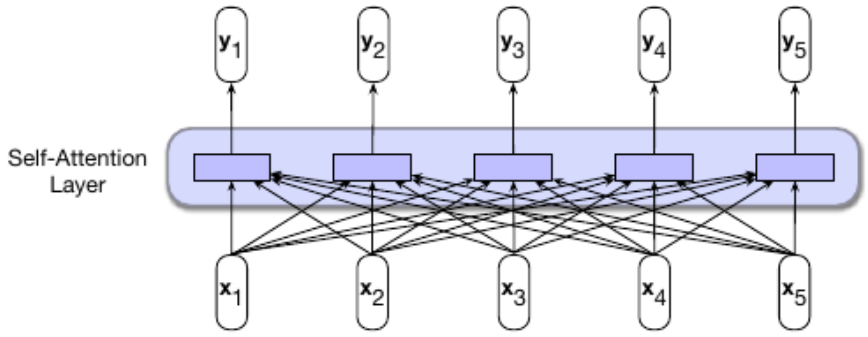
Las grandes variaciones de escala en las imágenes con una alta resolución espacial no se compara con lo que sería el texto y las palabras.



Shifted window. En cada ventana aplicamos self-attention. Notar que las ventanas son jerárquicas con distintas resoluciones.

Liu et al , 2021. Swin transformer: Hierarchical vision transformer using shifted windows. Proceedings IEEE/CVF.

Bidirectional Encoder Representations from Transformers (BERT)



Bidirectional encoder:

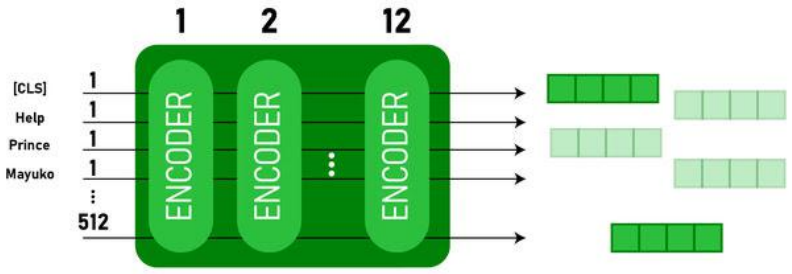
Proponen un codificador bidirectional.

No le aplican la máscara triangular.

Permite contextualizar cada token con la información de toda la oración.

- Las entradas son segmentadas usando tokenization subpalabra.
- Se combinan con embeddings posicionales.
- A éstos se los pasa por bloques transformers con self-attention y FCNN con conexiones residuales y capa de normalización.
- Para Seq2Seq se agrega decoder donde la matriz de atención es una máscara triangular.

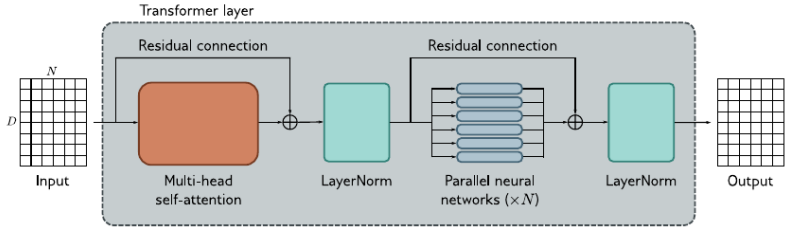
Embeddings en BERT



- Vocabulario de 30.000 tokens usando el algoritmo Word-Piece (sub-palabras!).
- Se trabaja con una longitud fija de 512 tokens de entrada.
- Los tokens de entrada son incrustados en un 1024-dimensional espacio.

- BERT tiene contextual embeddings el significado de la palabra va a depender del contexto.
- la misma palabra en distintos contextos tiene distintos embeddings.
- Las posiciones son consideradas a través del índice de la posición, con la misma dimension de los tokens (uno p/c token).

Arquitectura BERT



- Los vectores de entrada son pasados a 12/24 bloques de transformers (Bert-base).
- La cantidad de capas internas es de 768/1024.
- Tiene una sola FCNN de 4096 que esta conectada a la salida de los transformers.
- Tamaño del batch 256. Optimizador: ADAM
- Total de 110/340 millones de parámetros (una bicoca al lado de los 1.76 trillones de GPT4)

Codificación posicional

Position embeddings (PEs) son esenciales para capturar el orden de las palabras en los transformers sino serían BOW.
Posiciones absolutas (PA): posición absoluta del token en la oración.
Posiciones relativas (PR): posición de cada token con respecto a los otros tokens.
Variantes:

1. PA Aprendible PA
2. PA sinusoidal fijo.
3. PR aprendible.
4. PR sinusoidal fijo.

$$PE_{j,2i} = \sin(j/10^{4(2i/d_{mod})}); \qquad j \text{ posicion}, i \text{ dimension}, d_{mod} \text{ dimension de salida}$$

Wang, B., et al 2020: On position embeddings in bert. ICLR.

Pre-entrenamiento de BERT

Los parámetros del transformer de BERT son entrenados con auto-supervisión usando un gran corpus.

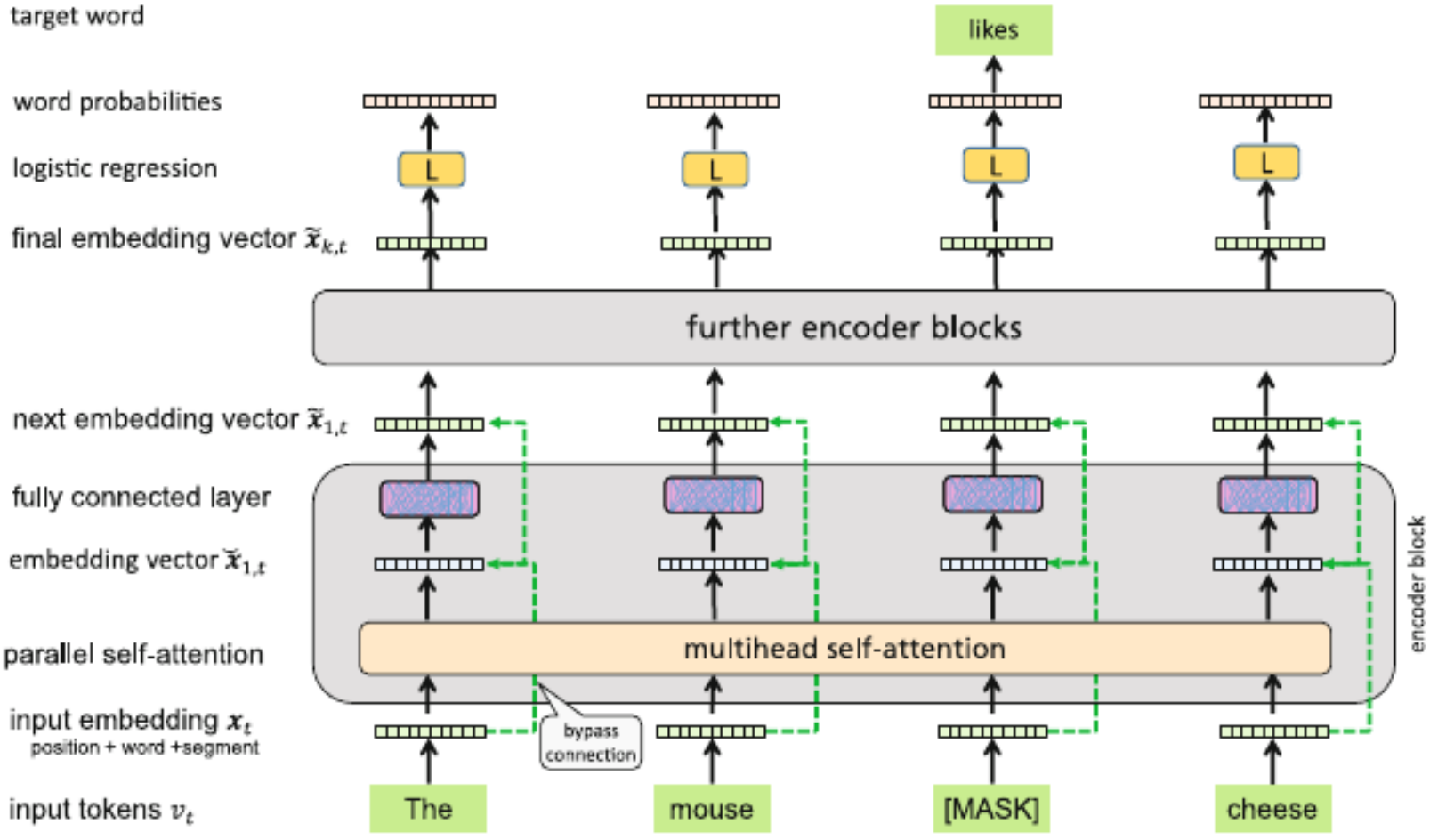
- Corpus: English Wikipedia y libros con 3.3 billones de palabras.
- Auto-supervisión Se predicen palabras de la oraciones las cuales son conocidas y pueden ser usadas como targets.
- La cantidad máxima de tokens de la oración es de 512. El token [CLS] ("clasificación") se pone al comienzo de cada oración de entrada.
- El 15% de los tokens de entrada son seleccionados para la predicción, usando el token [MASK] en todos los documentos del entrenamiento.

Ej. "[CLS] el señorial chalet se [MASK] a la izquierda de la escuela"
Un clasificador logístico estima la probabilidad de la máscara:

$$p(v_t|v_{1:t-1}, v_{t+1:T}) = \text{softmax}(\mathbf{A}\tilde{\mathbf{x}}_{k,t} + \mathbf{b})$$

BERT también predice si la próxima oración esta relacionada a la actual. Predicción de la oración siguiente. Función de pérdida multiobjetivo. De menos impacto en el entrenamiento.

Inferencia en BERT y su paralelismo



Sintonizado fino en BERT

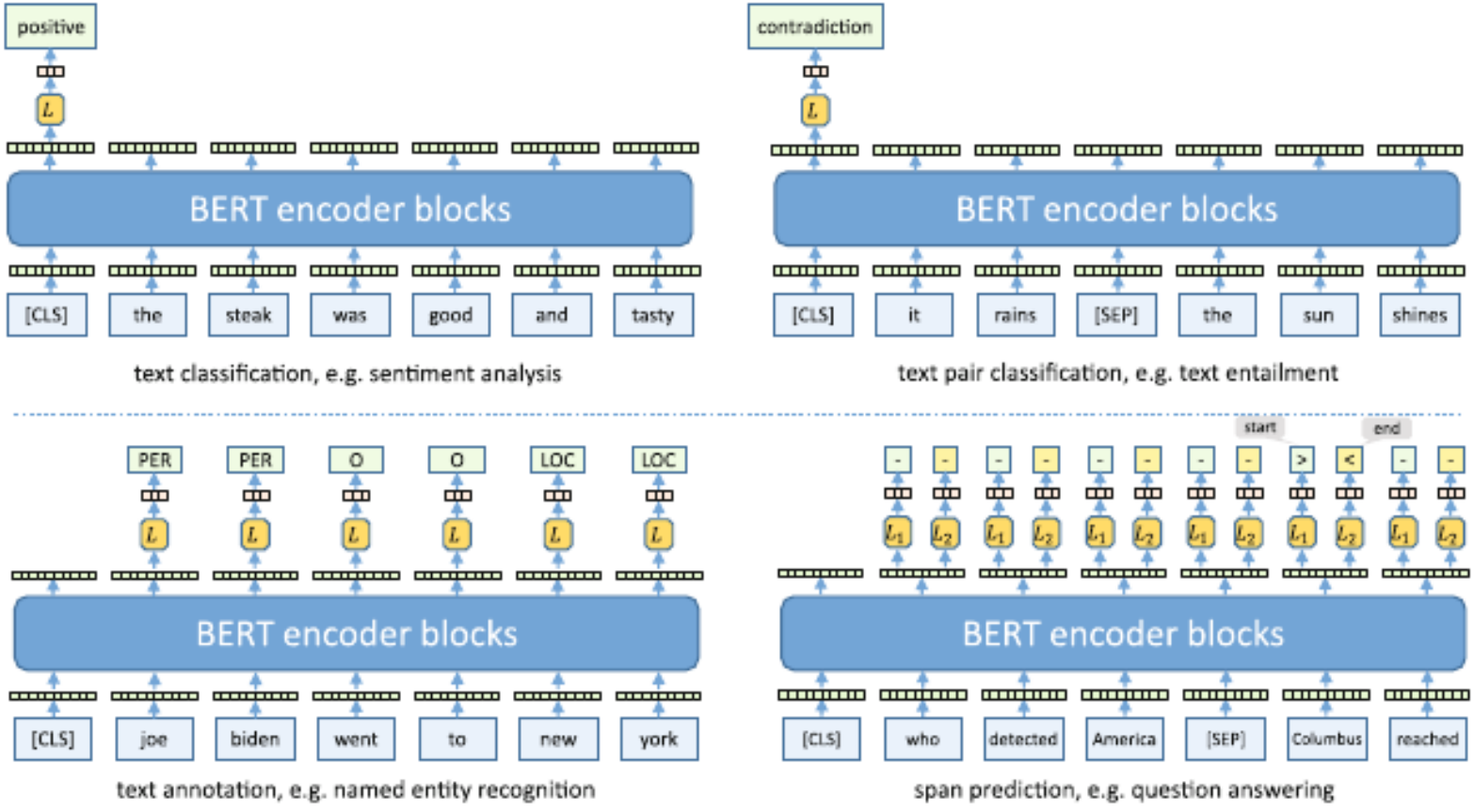
Con el pre-entrenamiento de BERT se aprende las propiedades sintácticas y semánticas del lenguaje.
Luego queremos realizar una aplicación específica:

- clasificación del texto. Análisis de sentimiento. Reclamos. Etc.
- clasificación de palabras. Sintaxis.
- preguntas y respuestas. Chatbots.

Para esto necesitamos hacer un sintonizado fino, vamos a hacer un transfer learning del preentrenamiento hacia una aplicación mas específica.
La red se adapta para la aplicación específica y el dataset a utilizar para el sintonizado es mucho mas pequeño.

En general se agrega una capa extra a los transformers para convertir los vectores de salida a la predicción requerida.
$\Rightarrow$ En el caso de clasificación, se pone una logística a la salida.

Sintonizado fino en BERT



Clasificación de texto. Definimos la clase. La salida del token [CLS] es usada para la fn de pérdida.

Clasificación de pares de sentencias. Separadas por [SEP].

Name entity recognition (NER). Extracción de información. Identificación de entidades

Modelado de tópicos con BERT

¿De que se esta hablando ahora en X? Cuales son las tendencias? Como hacemos para saber de documentos o tweets que hablan de lo mismo? $\rightarrow$ clustering

- Document/Paragraph embedding con BERT.
- Como el problema de la dimensionalidad no permite clustering. Primero reducimos la dimensionalidad: T-NSE o UMAP.
- Clustering con HDBSCAN. Permite que haya outliers que no pertenecen a ningun cluster.
- Finalmente para cada cluster aplicamos cTF-IDF. TF-IDF para cada cluster por separado.

Grootendorst, M., 2022. BERTopic: Neural topic modeling with a class-based TF-IDF procedure. arXiv preprint arXiv:2203.05794. https://arxiv.org/pdf/2203.05794.pdf

Modelado de tópicos con BERT

Modelado de tópico de las noticias de un diario. Palabras mas frecuentes (TF-IDF) sobre el cluster.

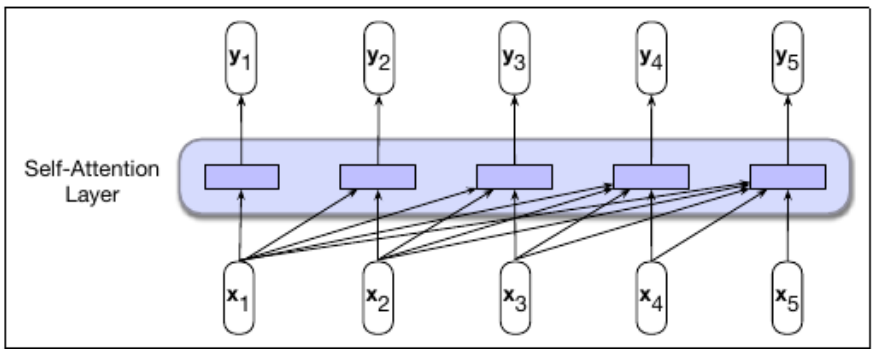
GPT

Modelo autoregresivo.

$$p(w_{1:n}) = p(w_1)p(w_2|w_1)\dots p(w_n|w_{1:n-1}) = \prod_{k=1}^{n} p(w_k|w_{k-1})$$

- Predicción del próximo token dados todos los anteriores (Conocidos).
- Se usan los transformers solo como un decoder

Usa self-attention para inferir los embeddings contextuales de los tokens pasados $w_{1:n-1}$ y luego predice el w_n.
La estructura es similar a la de BERT.
Usa embeddings posicionales tambien.



Causal self-attention

Los embeddings estan limitados al pasado, enmascara todas las palabras futuras en el self-attention.

$$p(w_{n+1}|w_{1:n}) = \text{softmax}(A\tilde{\mathbf{x}}_{k,t} + \mathbf{b})$$

Training GPT

La optimización se realiza a través de estimación por máximo verosimilitud:

$$J = -\log p(w_{n+1}|w_{1:n}) = -\log p(w_1) - \log p(w_2|w_1) - \cdots - \log p(w_n|w_{1:n-1})$$

- Teacher forcing. Usa toda la historia correcta $w_{1:n}$ para predecir el w_{n+1}.
- CommonCrawl corpora: WebText dataset (scraping links), internet books, English wikipedia.
- Ruido del gradiente usado para determinar tamaño del batch.
- GPT2. Input 1024 tokens. Tamaño del embedding 1024. 24 capas con 12 bloques de atención. Batch 512.
- GPT3. 175 millones de parametros

Link guía de uso

Generación de secuencias de palabras

¿Como hacemos para generar una oración?

- Muestreo aleatorio. Usando las probabilidades se generan muestras de cada palabra siguiente. No sirve porque aparecen palabras poco probables.
- Muestreo de los k-probables. Toma los k tokens de mayor probabilidad y luego muestrea entre éstos.
- p-cumulativa. Toma los candidatos que estan dentro de un umbral de la probabilidad cumulativa (0.95). Se redistribuyen las probabilidades en cada tiempo.

Retroalimentación humana en GPT

Se define el retorno a través de una NN con preferencias humanas entre pares de segmentos de las trayectorias.
En general es una preferencia. Cual esta mejor?
Requiere la opiniones humanas de solo el 1% de las interacciones con el entorno.
En lugar de pensar que el entorno tiene una respuesta del retorno, asumimos que hay un humano que manifiesta preferencias.
En cada tiempo de la trayectoria tenemos la política π y la función del retorno r parametrizada por deep NNs. Pasos:

- La política π interactua con el entorno y produce las trayectorias $\tau_{1:i}$. Los parámetros de π se renuevan con RL para maximizar el $r_t(o_t, a_t)$.
- Se seleccionan pares de segmentos de las trayectorias (σ_1, σ_2) y se las manda a un humano para evaluar.
- Los parámetros de $r(o_t, a_t)$ se optimizan con aprendizaje supervizado usando las medidas humanas.

Christiano, et al., 2017. Deep reinforcement learning from human preferences. NIPS, 30.

Queremos un modelo mas creativo. La temperatura

Tenemos que recordar que los modelos de language son un modelo probabilístico.
La salida es una sigmoide que nos da la probabilidad de cada token.

¿Qué token seleccionamos? El mas probable? Uno de los mas probables?
La temperatura controla la random-cidad ajustando la distribución de probabilidad del próximo token.

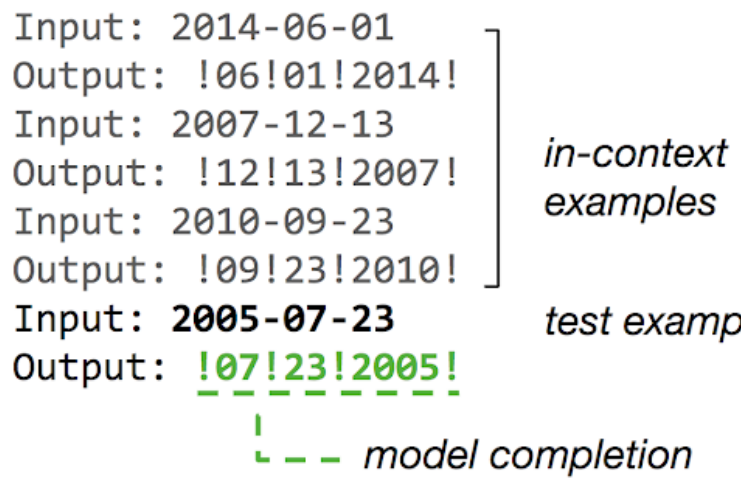
- 1. Valor por default. Balance entre generación y realismo.
- 0.2-0.5 baja temperatura. Tareas determinísticas (ej. resúmenes, respuestas, extracción de entidades).
- 1.2-2 alta temperatura. Poesía. Historias. Brainstorming.

In-context learning

Dentro de las estrategias de prompting el ICL realiza la pregunta en conjunto con demostración de contexto.

ICL no hace ningun tipo de cambios en los parámetros del modelo, solo la predicción con el modelo pre-entrenado.

- Se espera que el modelo aprenda del patron escondido en los ejemplos.
- Aprender por analogía.
- El ICL no requiere de fine-tuning.
- El impacto es notable (capacidad emergente) pero solo en grandes modelos de leguaje >50 billones.



¿Porque aprenden los LLMs de ICL?

- Language Models Implicitly Perform Gradient Descent as Meta-Optimizers
- In-context learning as Implicit Bayesian Inference

Change of thought learning - COT

Standard Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain-of-Thought Prompting

Model Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

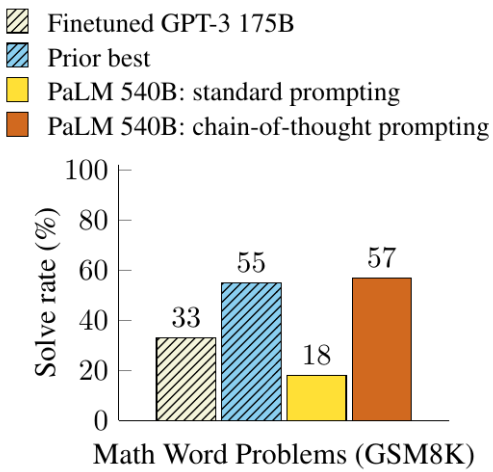
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓

Chain-of-Thought Prompting Elicits Reasoning in Large Language Models



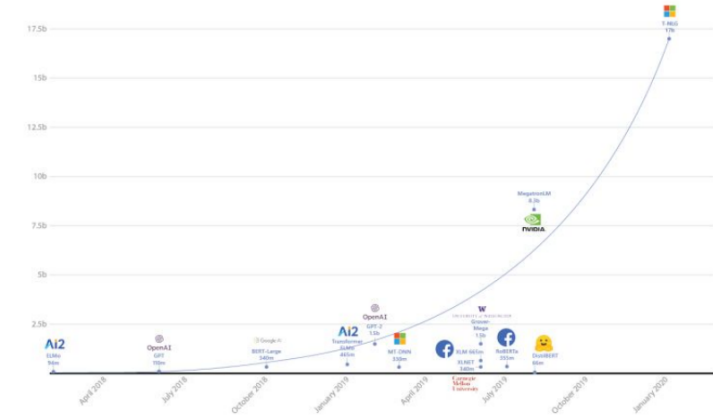
Lo podemos obligar a que razone paso a paso.

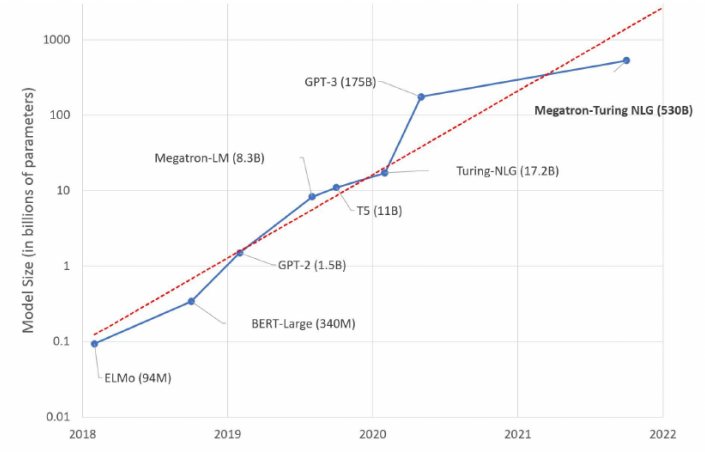
Let us think step by step

Large Language Models are Zero-shot Reasoners

Scaling law

Estudiar las relaciones de performance optima de acuerdo a los atributos principales: numero de parametros, cantidad de datos de entrenamiento, costo de computo para entrenamiento.





Inicialmente el paradigma fue aumentar la cantidad de parametros y entrenar modelos mas grande dada una base de datos de entrenamiento.

Scaling law: Nuevos paradigmas

Chinchilla (2023), un modelo 70B con una cantida de 1.4 trillion tokens, acorde a la ley 20:1.
Chinchilla 70B tuvo mejor performance que: Gopher (280B) GPT-3 (175B) Megatron-Turing NLG (530B)
$\Rightarrow$ Demuestra la importancia de equilibrar el tamaño del modelo con el volumen de datos de entrenamiento.
Llama-3 tiene una relación de 200:1 de token-to-parameter (y la ley 20:1?).

RAG - Retrieval Augmented Generation

La idea es combinar:

- un modelo para obtener información de documentos
- un modelo generativo

Modelo de retrieval

- busca dinámicamente documentos o información para proveer un contexto para pasar al modelo generativo.
- utiliza similarity search (FAISS) o búsquedas basadas en palabras claves (ElasticSearch)

Ventajas

- Permite utilizar base de datos específicas.
- Mejora la precisión de las respuestas.
- Disminuye la alucinación brindando un contexto específico.

Fin (Como humanidad el comienzo de nuestro...?)