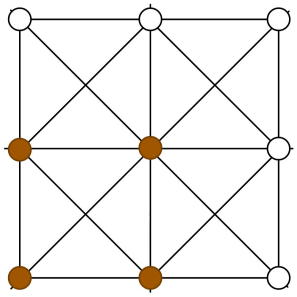
Facultad de Ciencias

El cap set problem: de la combinatoria al uso de LLM en matemática teórica.



Grado de Matemáticas

Autor: Daniel Expósito Viana

Tutor: Roberto Rubio

Junio 2025

1. Cap set problem

1.1. Introducción al cap set problem

Recientemente, el cap set problem ha emergido como un caso fascinante de interacción entre la matemática teórica y la inteligencia artificial. El avance vertiginoso de los Large Language Models ofrece aplicaciones nuevas de forma continua, y este problema combinatorio revela el potencial de la IA como herramienta matemática. Pero, ¿en qué consiste exactamente el cap set problem y por qué es relevante?

El cap set problem es una pregunta de combinatoria extremal. Originalmente se planteó en el contexto de geometría finita, pero su interés trasciende de esta área, siendo notable también en combinatoria aditiva y teoría de la información. Explicado brevemente, dado $n \in \mathbb{N}$, consiste en encontrar el subconjunto más grande de puntos de $(\mathbb{Z}/3\mathbb{Z})^n$ que no contenga tres puntos x, y, z tales que $x + y + z = 0$. Desde el prisma de la combinatoria aditiva, esto es equivalente a que x, y, z formen una *progresión aritmética* de 3 términos. Una progresión aritmética es una secuencia donde la diferencia entre términos consecutivos es constante. Formalmente, tres puntos que satisfacen $y - x = z - y$, lo que implica $x + z = 2y$, que en $\mathbb{Z}/3\mathbb{Z}$ se puede reescribir como $x + y + z = 2y + y = 0$.

La primera vez que se propuso este problema fue en 1947, cuando Bose investigó el tamaño máximo de los *caps*, definidos como subconjuntos de la geometría proyectiva $\mathbb{P}(\mathbb{F}_q^n)$ sobre el cuerpo finito $\mathbb{F}_q^n$ que no contienen tres puntos colineales [Gro18].

En el campo de la combinatoria aditiva, donde más recorrido ha tenido, fue el trabajo de Roth el que inició la línea teórica que acabó desembocando en el cap set problem. En 1953, prueba que el subconjunto más grande de $[N] := \{1, \ldots, N\}$ que no contiene progresiones aritméticas de 3 términos es de tamaño $o(N)$,[1] de hecho, de la investigación de Roth se puede deducir una

[1] La notación $o(N)$ describe una función que f tal que $\lim_{N\to\infty} \frac{f(N)}{N} = 0$.

cota de la forma $O(N/\log\log N)$ [CLP24].[2] Este avance motivó una pregunta más, conocida como el problema de Roth: ¿cuál es el tamaño máximo de un subconjunto de $[N]$ que no contiene progresiones aritméticas de 3 términos? No fue hasta 1995, que Meshulam reformuló el problema de Roth en $(\mathbb{Z}/3\mathbb{Z})^n$ [Mes95].

La investigación del cap set problem ha impulsado avances en diversas áreas del conocimiento matemático. En 2016, Ellenberg y Gijswijt [EG16] desarrollaron una técnica conocida como *slice rank*, que fue clave en progresos de problemas abiertos como la conjetura del sunflower, un problema relacionado con la intersección estructurada de familias de conjuntos. En el contexto de la programación, las cotas superiores del cap set implican cotas inferiores en ciertos algoritmos de multiplicación de matrices. En geometría discreta, mejoró cotas para el problema de centroides enteros de Harborth para el caso de dimensión 3 [Gro18].

En resumen, el impacto del cap set problem ha trascendido su formulación original, consolidándose como un problema central cuyas soluciones se han proyectado en múltiples ramas de las matemáticas. Este fenómeno refleja cómo una pregunta aparentemente específica puede actuar como catalizador de nuevas técnicas y conexiones interdisciplinarias, incluyendo, como se verá más adelante, el papel creciente de los modelos de lenguaje en la búsqueda automatizada de estructuras combinatorias óptimas.

Contribuciones: En este trabajo hemos llevado a cabo un estudio profundo sobre el cap set problem, con el objetivo de comprender su formulación, evolución histórica y los avances más significativos. Aunque este trabajo no presentamos resultados originales en términos de demostraciones o avances propios, ofrecemos una recopilación estructurada y comprensible de tres componentes fundamentales para la interpretación del reciente abordaje computacional del cap set problem. El trabajo está separado en tres secciones.

[2]Una función $f(N)$ es de la forma $O\left(\frac{N}{\log\log N}\right)$ si existe una constante $C > 0$ tal que $f(N) \leq C \cdot \frac{N}{\log\log N}$ para todo N suficientemente grande.

En la primera sección, exponemos el marco teórico que sustenta la búsqueda de cotas inferiores para el cap set problem y las estrategias constructivas utilizadas.

En la segunda parte del trabajo, introducimos los fundamentos matemáticos detrás de la arquitectura de los transformers, en la que se basan large language models (LLM) actuales, como PaLM. Este análisis ha sido clave para comprender la tercera y última parte del trabajo, centrada en el artículo de Nature sobre FunSearch, donde se combinan large language models con algoritmos evolutivos para tratar el cap set problem.

Intentamos aplicar una estructura análoga a la de FunSearch al problema propuesto por Harborth en 1973. Harborth introdujo la función $s(m, n)$, definida como el mínimo número s tal que cualesquiera s puntos en $\mathbb{Z}^n$ contiene un subconjunto de m elementos cuyo centroide tiene también coordenadas enteras. Que se puede reformular equivalentemente como el mínimo s tal que cualquier sucesión de s elementos en $\mathbb{F}_3^n$ contiene una subsucesión de longitud m cuya suma es cero. Para $m = 3$, tenemos un caso muy parecido al cap set problem, con la diferencia de que se pueden formar sucesiones de elementos repetidos [Gro18]. Esta similitud motivó nuestro intento de adaptar el esqueleto de FunSearch al nuevo marco, incorporando restricciones adicionales específicas del problema de centroides. Sin embargo, no fue posible implementarlo debido a las dificultades encontradas al manipular y comprender las complejas estructuras de programación necesarias, así como al elevado requerimiento computacional que implica la ejecución del algoritmo evolutivo y el LLM.

El trabajo ha implicado una investigación profunda que puede servir como punto de partida para futuros trabajos orientados a la investigación teórica del cap set o desarrollo de herramientas computacionales aplicadas.

1.2. Marco teórico

Definición 1.1 (Cap set)**.** *Un cap set es un conjunto $A \subseteq \mathbb{F}_3^n$ sin soluciones a $x + y + z = 0$ excepto cuando $x = y = z$, o equivalentemente un conjunto A sin 3 elementos diferentes que formen una progresión aritmética.*

Comencemos con un poco de notación. Por comodidad, utilizaremos $\mathbb{F}_3^n$ en vez de $(\mathbb{Z}/3\mathbb{Z})^n$.

Problema 1 (Cap Set Problem)**.** *Determinar el tamaño máximo de un cap set $A \subseteq \mathbb{F}_3^n$ a medida que aumenta n.*

Definimos $r_3(Z)$ como el tamaño del subconjunto más grande de Z sin progresiones aritméticas de 3 términos, a nosotros nos interesa $r_3(\mathbb{F}_3^n)$.

La historia de este problema es todavía reciente y breve, pero en los últimos años se han concentrado avances importantes. Dividiré la historia en dos secciones, la cota superior y la cota inferior, en la que profundizaré más. Hago esta separación porque los enfoques para avanzar en cada problema son muy diferentes. Para estudiar la cota superior, tradicionalmente se ha utilizado análisis de Fourier y en el campo de la cota inferior se han usado métodos constructivos y combinatorios.

1.2.1. Cota superior

Si bien la pregunta ya se planteaba en 1993 [AD93], la siguiente conjetura no se propuso formalmente hasta 2004.

Conjetura 1.2 (Conjetura del Cap Set [Gre04])**.** *Existe una constante absoluta $\delta > 0$ tal que $r_3((\mathbb{F}_3^n) \leq (3 - \delta)^n$.*

Su resolución en 2016 fue una sorpresa debido a que ni siquiera existía un consenso acerca de qué método se tenía que seguir para llegar a demostrarla. Pero comencemos con las primeras cotas superiores que se dieron.

Aunque existen trabajos anteriores, como el de Roth en 1953 y Szemerédi en 1975 [Sze75], el primer artículo en tratar el problema en $\mathbb{F}_3^n$ fue el de

Brown y Buhler [BB82]. En esta primera aproximación usaron una versión geométrica del teorema de Ramsey para demostrar que los conjuntos sin progresiones de tres términos en $\mathbb{F}_3^n$ no pueden constituir una proporción constante del espacio total, es decir, su densidad tiende a cero conforme n crece. Esto implica directamente que $r_3(\mathbb{F}_3^n)$ tiene tamaño $o(3^n)$.

La primera cota superior explícita fue dada por Meshulam en 1995 [Mes95]. Meshulam adaptó la técnica de Roth basada en análisis de Fourier (originalmente sobre $\mathbb{Z}$) al contexto de espacios vectoriales finitos. Aplicando el Teorema 1.2 del ensayo [Mes95] a $\mathbb{F}_3^n$ se concluye que $r_3(\mathbb{F}_3^n) \leq 2 \cdot \frac{3^n}{n}$.

El siguiente avance en la cota superior vino en 2011. Bateman y Katz demostraron que $r_3(\mathbb{F}_3^n) \leq \frac{3^n}{n^{1+\varepsilon}}$ para $\varepsilon > 0$ pequeño. Introdujeron nuevas herramientas de análisis para la combinatoria aditiva con las que prepararon las bases que impulsaron el desarrollo del método polinomial [Gro18].

En 2016, Croot, Lev y Pach [CLP16] introdujeron una nueva forma de usar el método polinomial, dejando atrás el análisis de Fourier para tratar el problema. En su caso, mejoraron el resultado de Sanders sobre $r_3((\mathbb{Z}/4\mathbb{Z})^n)$, que consiguieron acotar superiormente por c^n para $c < 4$. Este resultado, tan parecido a la Conjetura 1.2, propulsó su resolución tan solo 25 días después. Ellenberg y Gijswijt utilizan el método polinomial aplicado a $\mathbb{F}_3^n$ para demostrar finalmente que $r_3(\mathbb{F}_3^n) < c^n$ para un $c < 3$. Por último, en 2023, Zhi Jiang refinó el método de Ellenberg y Gijswijt para mejorar la cota por un factor de $\sqrt{n}$.

1.2.2. Cota inferior

A diferencia de en la cota superior, donde los avances venían dados por métodos diferentes cada vez, la investigación de la cota inferior se basa en la construcción de conjuntos maximales de $\mathbb{F}_3^n$ sin progresiones aritméticas de tres términos en dimensiones pequeñas y, seguidamente, combinarlos inteligentemente para producir cap sets de dimensiones superiores.

La primera cota inferior exponencial del cap set problem vino del trabajo

de Calderbank y Fishburn en 1970. Consiguieron una cota de $(2{,}2101147...)^n$. No fue hasta 2004 que Edel [Ede04] mejoró esta cota hasta $(2{,}217389...)^n$ basándose en el trabajo de Calderbank y Fishburn. Finalmente, el último avance "humano"de esta cota es el de Tyrrell [Tyr22] que, siguiendo la metodología de Edel e implementando herramientas computacionales mejoradas, consiguió una cota inferior de $(2{,}218021...)^n$.

A partir de aquí, seguiremos brevemente el articulo de Tyrrell [Tyr22] para entender la linea de trabajo con la que se consiguieron mejores resultados antes de los Large Language Models.

Empezaré explicando la construcción de producto extendido de Edel y luego la construcción introducida por Tyrrell, que mejora la anterior. Y por último los métodos computacionales que le llevaron al resultado.

Proposición 1.3 (Product caps)**.** *Sea $A \subseteq \mathbb{F}_3^n, B \subseteq \mathbb{F}_3^m$ cap sets. Entonces, considerando el producto directo de A y B, hay un cap set de tamaño $|A||B|$ en $\mathbb{F}_3^{n+m}$*

Demostración. Por definición $A \times B = \{(a, b) : a \in A, b \in B\}$. Y por tanto, $|A \times B| = |A||B|$. Supongamos que tenemos tres elementos diferentes del producto directo $x = (a_1, b_1), y = (a_2, b_2), z = (a_3, b_3)$ con $a_1, a_2, a_3 \in A$ y $b_1, b_2, b_3 \in B$. Si al menos un elemento a_1, a_2, a_3 es diferente de los otros, como A es un cap set, implica que $x + y + z \neq 0$. Si $a_1 = a_2 = a_3$, como hemos escogido elementos diferentes del producto directo, b_1, b_2, b_3 son diferentes entre sí, y de nuevo, como B de cap set, $x + y + z \neq 0$. □

Ahora, usando la construcción anterior veamos cómo podemos deducir una cota inferior asintótica para el cap set problem.

Proposición 1.4. *Sea $A \subseteq \mathbb{F}_3^n$ un cap set de tamaño c^n. Entonces, para todo $\varepsilon > 0$, existe un M tal que para todo $m > M$, hay un cap set con un tamaño más grande que $(c - \varepsilon)^m$ en $\mathbb{F}_3^m$*

Demostración. Sea $A \subseteq \mathbb{F}_3^n$ un cap set de tamaño c^n. Para $m > n$ existe k tal que $m = kn + r$, donde $0 \leq r \leq n$. Aplicando la construcción del producto directo k veces sobre A, tenemos un cap set de tamaño $c^{kn} = \left(c^{1-r/m}\right)^m$.

Dado un $\varepsilon > 0$, podemos escoger un M tal que $c^{1-n/M} > c - \varepsilon$. Como $r < n$ y $m \geq M$ tenemos que $c^{1-r/m} > c - \varepsilon$. Por tanto, podemos construir un cap set en $\mathbb{F}_3^m$ con tamaño más grande que $(c - \varepsilon)^m$ □

Observación. *Es equivalente encontrar una cota inferior asintótica para el cap set problem a encontrar un cap set $A \subseteq \mathbb{F}_3^n$ tal que $|A|^{1/n}$ sea lo más grande posible. Sean $A \subseteq \mathbb{F}_3^n$ y $B \subseteq \mathbb{F}_3^m$ cap sets, su producto directo tendrá un tamaño $|A \times B|^{1/(m+n)}$, que es menor o igual a max$(|A|^{1/n}, |B|^{1/m})$ y por tanto la cota obtenida a partir del producto directo nunca superará a la mejor cota de A y B.*

La idea detrás de las siguientes definiciones es construir una serie de productos directos a partir de una colección de cap sets tales que, al considerar su unión, siga siendo un cap set.

Definición 1.5 (Colección extensible)**.** *Sea $A_0, A_1, A_2 \subseteq \mathbb{F}_3^n$ cap sets. Decimos que esta colección de cap sets es extensible si se cumplen las siguientes dos condiciones:*

1. *Si $x, y \in A_0$ y $z \in A_1 \cup A_2$, entonces $x + y + z \neq 0$.*
2. *Si $x \in A_0$, $y \in A_1$ y $z \in A_2$, entonces $x + y + z \neq 0$.*

Nótese que tomar $x = y$ en la condición (1) implica que A_0 es disjunto de A_1 y A_2.

Definición 1.6 (Conjunto admisible)**.** *Sea $S \subseteq \mathbb{F}_3^n$. Decimos que S es admisible si se cumplen las siguientes condiciones:*

1. *Para todos los elementos distintos $s, s' \in S$, existen coordenadas i y j tales que:*

$$s_i = 0 \neq s'_i \quad y \quad s_j \neq 0 = s'_j.$$

2. *Para todos los elementos distintos $s, s', s'' \in S$, existe una coordenada k tal que:*

$$\{s_k, s'_k, s''_k\} = \{0,1,2\}, \quad \textit{o bien } \{0,0,1\}, \quad \textit{o bien } \{0,0,2\}.$$

Definición 1.7 (Construcción de producto extendido)**.** *Como sugiere el nombre, podemos extender una colección extensible de cap sets mediante un conjunto admisible. La construcción es la siguiente:*

Sean $s = (s_1, \ldots, s_m) \in \mathbb{F}_3^m$ y $A_0, A_1, A_2 \subseteq \mathbb{F}_3^n$. Definimos:

$$s(A_0, A_1, A_2) = A_{s_1} \times \cdots \times A_{s_m} \subseteq \mathbb{F}_3^{nm}.$$

Si $S \subseteq \mathbb{F}_3^m$ es un conjunto admisible, definimos:

$$S(A_0, A_1, A_2) = \bigcup_{s \in S} s(A_0, A_1, A_2) \subseteq \mathbb{F}_3^{nm}.$$

Veamos ahora la motivación de estas definiciones.

Lema 1.8. *Si (A_0, A_1, A_2) es una colección extensible de cap set en $\mathbb{F}_3^n$, y si $S \subseteq \mathbb{F}_3^m$ es un conjunto admisible, entonces $S(A_0, A_1, A_2)$ es un cap set en $\mathbb{F}_3^{nm}$.*

Demostración. Queremos demostrar que

$$\bigcup_{s \in S} s(A_0, A_1, A_2)$$

es un *cap set*, donde $s(A_0, A_1, A_2) = A_{s_1} \times \cdots \times A_{s_m}$.

Supongamos que tenemos elementos distintos $x, y, z \in S(A_0, A_1, A_2)$ tales que $x + y + z = 0$. Consideramos tres casos, dependiendo del origen de x, y, z:

Caso 1: Por la proposición 1.3, sabemos que cada $s(A_0, A_1, A_2)$ es un cap set. Por lo tanto, no existen elementos distintos $x, y, z \in s(A_0, A_1, A_2)$ tales que $x + y + z = 0$.

Caso 2: Supongamos que $x, y \in s(A_0, A_1, A_2)$ y $z \in s'(A_0, A_1, A_2)$, con $s \neq s'$. Entonces $x = (x_{s_1}, \ldots, x_{s_m})$, $y = (y_{s_1}, \ldots, y_{s_m})$, y $z = (z_{s'_1}, \ldots, z_{s'_m})$. Por la propiedad (1) de ser un conjunto admisible, existe una coordenada j tal que $s_j = 0$ y $s'_j \neq 0$. Entonces $x_{s_j} + y_{s_j} + z_{s'_j} = 0$, donde $x_{s_j}, y_{s_j} \in A_0$ y $z_{s'_j} \in A_1 \cup A_2$, lo cual contradice la propiedad (1) de ser extendible.

Caso 3: Supongamos que x, y, z provienen de vectores distintos s, s', s''. Por la condición (2) de ser un conjunto admisible, existe una coordenada k tal que

$$\{s_k, s'_k, s''_k\} = \{0, 1, 2\}, \quad \text{o bien } \{0, 0, 1\}, \quad \text{o bien } \{0, 0, 2\}.$$

Si el conjunto es $\{0, 0, 1\}$ o $\{0, 0, 2\}$ podemos utilizar el mismo argumento que en el caso anterior, llegando a contradicción por la propiedad (1) de extendible. Si el conjunto es $\{0, 1, 2\}$, entonces tenemos una contradicción con la propiedad (2) de extendible. Ya que, reordenando si hace falta, $x_{s_k} \in A_0$, $y_{s_k} \in A_1$ y $z_{s_k} \in A_2$, entonces $x_{s_k} + y_{s_k} + z_{s_k} \neq 0$.

□

Definición 1.9 (Conjunto recursivamente admisible)**.** *Un conjunto $S \subseteq \mathbb{F}_3^m$ es un conjunto recursivamente admisible si:*

- *S es un conjunto admisible,*
- *$|S| \geq 2$,*
- *y para todo par distinto $s, s' \in S$, se cumple al menos una de las siguientes condiciones:*

  (i) Existen coordenadas i y j tales que:

  $$\{s_i, s'_i\} = \{0, 1\} \quad y \quad \{s_j, s'_j\} = \{0, 2\}.$$

  (ii) Existe una coordenada k tal que $s_k = s'_k = 0$.

Lema 1.10. *Si (A_0, A_1, A_2) es una colección extensible de cap sets, y $S \subseteq \mathbb{F}_3^m$ es un conjunto recursivamente admisible, entonces:*

$$(S(A_0, A_1, A_2), A_1^m, A_2^m)$$

es una colección extensible de cap sets.

La demostración de este lema es parecida a la anterior, utilizando las definiciones anteriores.

Introduzcimos ahora el concepto de *peso* de un vector, que es el número de entradas diferentes de cero. También es útil conocer del *soporte* de un vector, es decir, el conjunto de coordenadas no nulas.

Definición 1.11 (Conjuntos admisibles de peso constante)**.** *Escribimos $S = I(m, w)$ si $S \subseteq \mathbb{F}_3^m$ es un conjunto admisible que consiste en $\binom{m}{w}$ vectores, cada uno con peso w. Si además S es recursivamente admisible, escribimos $S = \widetilde{I}(m, w)$.*

Una ventaja de esta clase de conjuntos admisibles es que satisfacen automáticamente la condición por pares de los conjuntos admisibles. Existen $\binom{m}{w}$ formas distintas de elegir w coordenadas de entre m para que sean distintas de cero, y entonces cualquier par de vectores distintos x, y necesariamente tendrá coordenadas i y j tales que $x_i = 0 \neq y_i$ y $x_j \neq 0 = y_j$.

Otra razón por la que este tipo de conjunto admisible es útil es que permite calcular el tamaño del cap set extendido de manera relativamente simple.

Lema 1.12. *Si extendemos una colección de cap sets (A_0, A_1, A_2) mediante $S = I(m, w) \subseteq \mathbb{F}_3^m$, donde $|A_1| = |A_2|$, entonces:*

$$|S(A_0, A_1, A_2)| = \binom{m}{w} |A_0|^{m-w} |A_1|^w.$$

Demostración. Sea $s \in S$. Si $s_i = 1$ o $s_i = 2$, entonces $|A_{s_i}| = |A_1|$, y si $s_i = 0$, entonces $|A_{s_i}| = |A_0|$. Como hay $m - w$ coordenadas cero y w coordenadas

distintas de cero en s, habrá $m - w$ conjuntos A_0 en el producto directo y el resto (w) serán A_1 o A_2, por tanto, $|s(A_0, A_1, A_2)| = |A_0|^{m-w}|A_1|^w$. Por último, como $S(A_0, A_1, A_2) = \bigcup_{s \in S} s(A_0, A_1, A_2)$, los conjuntos $s(A_0, A_1, A_2)$ son disjuntos entre sí y $|S| = \binom{m}{w}$, se sigue que:

$$|S(A_0, A_1, A_2)| = \binom{m}{w} |A_0|^{m-w}|A_1|^w.$$

□

Con esta base teórica, Edel construye un cap set aplicando el lema 1.10 con $S = \widetilde{I}(8, 1)$ (que existe por [Ede04, Lem. 13]), sobre la colección extensible de cap sets formada por, dos versiones del doubled Hill cap (un cap set maximal en $\mathbb{F}_3^6$) y un capset de 12 elementos en $\mathbb{F}_3^6$ (la construcción de la colección esta explicada en detalle en el apartado 3 de [Ede04]). Sobre al cap set resultante aplica el lema 1.8, con $S = I(10, 5)$, que construye en el apartado 4. Por último, aplica el Lema [Ede04, Lem. 11] para añadir puntos adicionales mediante la unión al conjunto anterior. El resultado es un cap set E en $\mathbb{F}_3^{480}$ con tamaño:

$$|E| = 32^{80} + 8^5 \cdot \binom{10}{5} \cdot 112^{75} \cdot 12^5$$

Y por la proposición 1.4 sabemos que que se traduce en una cota asintótica inferior de $r_3(\mathbb{F}_3^n) \geq 2{,}2173^n$ ya que, $|E|^{1/480} \approx 2{,}2173$.

Todo lo explicado hasta el momento es el trabajo de Edel [Ede04] que es estrictamente necesario para comprender las siguientes definiciones y resultados (por eso la construcción del cap set es tan poco minuciosa). Lo que viene a continuación es la extensión de este método con la que Tyrrell [Tyr22] llegó a la mejor cota inferior antes de la aparición de los Large Language Models. La idea que seguía Tyrrell era imitar la construcción del producto extendido de cap sets aplicado a conjuntos admisibles.

Proposición 1.13. *Si S, T son conjuntos admisibles, entonces su producto directo $S \times T$ también es un conjunto admisible.*

Demostración. Empecemos con la primera condición de admisible. Sea $s = (s_1, \ldots, s_m)$ un elemento de S y $t = (t_1, \ldots, t_n)$ un elemento de T, entonces un elemento de $S \times T$ tiene la forma $(s, t) \in \mathbb{F}_3^{m+n}$. Supongamos que tenemos dos elementos distintos (s, t) y (s', t') en $S \times T$. Como S y T son admisibles, si $s \neq s'$, entonces existen coordenadas i y j tales que $s_i = 0 \neq s'_i$ y $s_j \neq 0 = s'_j$. Estas diferencias las hereda el elemento del producto. Si en cambio $s = s'$ pero $t \neq t'$, entonces existen coordenadas i', j' tales que $t_{i'} = 0 \neq t'_{i'}$ y $t_{j'} \neq 0 = t'_{j'}$, y nuevamente las diferencias están presentes en las coordenadas del producto. Por tanto, la condición (1) de admisible se satisface.

Veamos ahora la segunda condición de admisible. Sean $a = (s, t)$, $b = (s', t')$ y $c = (s'', t'')$ elementos distintos de $S \times T$. Si s, s', s'' o t, t', t'' son todos distintos, entonces la condición (2) de admisibilidad se sigue directamente de la admisibilidad de S o T. Supongamos que ni s, s', s'' ni t, t', t'' son todos distintos. Sin pérdida de generalidad, podemos asumir que $s = s'$. Como $(s, t) \neq (s', t')$, no puede ocurrir que $t = t'$. Pero como t, t', t'' no son todos distintos, sin pérdida de generalidad podemos asumir que $t = t''$. Entonces tenemos $a = (s, t)$, $b = (s, t')$ y $c = (s'', t)$. Dado que a, b, c son distintos, debe cumplirse que $s \neq s''$. Por la condición (1) de admisibilidad de S, existe una coordenada k tal que $s_k = 0 \neq s''_k$. Entonces $\{a_k, b_k, c_k\} = \{0, 0, 1\}$ o $\{0, 0, 2\}$, por lo tanto también se cumple la condición (2) de admisibilidad para $S \times T$. □

Definición 1.14 (Meta-extensible)**.** *Una colección $S_0, S_1, S_2 \subseteq \mathbb{F}_3^m$ de conjuntos admisibles es meta-extensible si se cumplen las siguientes condiciones:*

1. *Todo $s \in S_0$ tiene menor peso (menos coordenadas no nulas) que cualquier $s' \in S_1 \cup S_2$.*

2. *Si $x, y \in S_0$ y $z \in S_1 \cup S_2$, entonces existe una coordenada k tal que $\{x_k, y_k, z_k\} = \{0, 1, 2\}$, $\{0, 0, 1\}$ o $\{0, 0, 2\}$.*

3. *Si $x \in S_0$, $y \in S_1$, $z \in S_2$, entonces existe una coordenada k tal que $\{x_k, y_k, z_k\} = \{0,1,2\}$, $\{0,0,1\}$ o $\{0,0,2\}$.*

Definición 1.15. *Si S_0, S_1, S_2 son conjuntos admisibles y $T \subseteq \mathbb{F}_3^r$, entonces para cada $t = (t_1, \ldots, t_r) \in T$ definimos:*

$$t(S_0, S_1, S_2) = S_{t_1} \times \cdots \times S_{t_r}.$$

Definimos luego el conjunto extendido:

$$T(S_0, S_1, S_2) = \bigcup_{t \in T} t(S_0, S_1, S_2).$$

Lema 1.16. *Si $S_0, S_1, S_2 \subseteq \mathbb{F}_3^m$ es una colección meta-extensible de conjuntos admisibles y $T \subseteq \mathbb{F}_3^r$ es admisible, entonces $T(S_0, S_1, S_2)$ es un conjunto admisible.*

No explico la demostración por su extensión y similitud a las anteriores, sin ser un resultado muy relevante más adelante.

Finalmente, lo que sigue es la aplicación de estos lemas y definiciones para construir un conjunto admisible que, con el mismo método que uso Edel para construir su cap set, nos de el resultado que buscamos, la cota de $2{,}218^n$

Primero, construye una colección de conjuntos admisibles meta-extensible, usa $S_1 = I(11, 7)$ y S_2 el mismo conjunto pero cambiando los 1s y 2s, y $S_0 \subseteq I(11, 3)$ y $|S_0| = 37$. Todos los conjuntos fueron encontrados usando el SAT solver que luego explicaré.

Ahora aplica el lema 1.16 con $T = I(142, 141)$, que existe por el [Ede04, Lem. 13], sobre la colección de conjuntos admisibles meta-extensible que hemos construido antes y tenemos un conjunto admisible R con $142 \cdot 37 \cdot \binom{11}{7}^{141}$ elementos. Finalmente, usando $S = \widetilde{I}(142, 141)$ (que existe también por el [Ede04, Lem. 13]), con el lema 1.10 sobre la misma colección de cap sets que usó Edel (dos versiones del doubled Hill cap y un cap set de 12 elementos en $\mathbb{F}_3^6$); y aplicando el lema 1.8 sobre el resultado con el conjunto admisible

R, obtenemos el cap set final. Este cap set en $\mathbb{F}_3^{56232}$ tiene tamaño $|A| \approx 2{,}21 \times 10^{19455} \approx 10^{104{,}3}$, y como $|A|^{1/56232} \approx 2{,}218$, esto proporciona una cota inferior exponencial de $2{,}218^n$.

Para encontrar los conjuntos admisibles se ha usado el kissat SAT solver. En corto, un solucionador de problemas de *boolean satisfiability*, que consisten en, dada una fórmula lógica, determinar si existe una asignación de valores, de verdadero o falso, a cada variable de la fórmula tal que la fórmula es verdadera. Cuando crearon la fórmula lógica que sirve de input para el SAT solver, decidieron añadir restricciones adicionales a las de ser admisible para reducir el tiempo de ejecución. Basándose en ejemplos y con prueba y error consiguieron nuevas restricciones que no hiciesen el la fórmula imposible de satisfacer. Por ejemplo, para crear $I(10, 6)$, añadieron que ningún vector acababa en (2,2), entre otras restricciones.

2. Redes neuronales

El objetivo de este apartado es explicar como se construyen y entrenan los Large Language Models (LLM), empezando desde las estructuras más básicas de machine learning hasta los transformers.

La construcción de un modelo de machine learning es, simplemente, crear una función de un espacio euclidiano a otro. Puede tener un estructura más o menos compleja, pero, el computo de sus partes acaba siendo eso. Como su nombre indica, nos ayudaremos de herramientas computacionales para afinar la función, pero eso lo veremos más adelante.

Definamos ahora la *arquitectura* básica de una red neuronal. Sea $G = (V, A)$ un grafo dirigido acíclico tal que el grado mínimo es al menos 1. Diferenciamos los vértices en tres conjuntos [BK24]:

- I el subconjunto de vértices que no reciben aristas; los llamaremos *inputs*.

- O el subconjunto de vértices que no emiten aristas; los llamaremos *outputs*.
- $H := V(I \cup O)$.

2.1. Perceptrón

El perceptrón es la estructura más básica sobre la que se construyen las redes neuronales; podemos pensar en él como una única neurona. Podemos ver la estructura en la Fig. 1. Sea $\mathbf{I} = \{x_1, x_2, ..., x_n\}$ el *conjunto de inputs*, intuitivamente, la información que recibe la neurona. Sean los parámetros $\{w_1, w_2, ..., w_n\}$ el *conjunto de pesos* y w_0 el sesgo. Definimos $\sigma : \mathbb{R} \to \mathbb{R}$ es la *función de activación no lineal*.

La *preactivación* z se define como

$$z = \sum_{i=1}^{n} x_i \cdot w_i$$

.

La *activación* $\hat{y}$ es el resultado de aplicar σ a la preactivación. La activación es el output del perceptrón [AA24].

$$\hat{y} = \sigma\left(w_0 + \sum_{i=1}^{n} x_i \cdot w_i\right)$$

Reescribiendo la fórmula usando álgebra lineal $\mathbf{I} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ $\mathbf{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}$

$$\hat{y} = \sigma\left(w_0 + \mathbf{I}^T\mathbf{W}\right)$$

El propósito de la función de activación es introducir la no linealidad a la red neuronal, haciéndola más expresiva con menos profundidad, y captu-

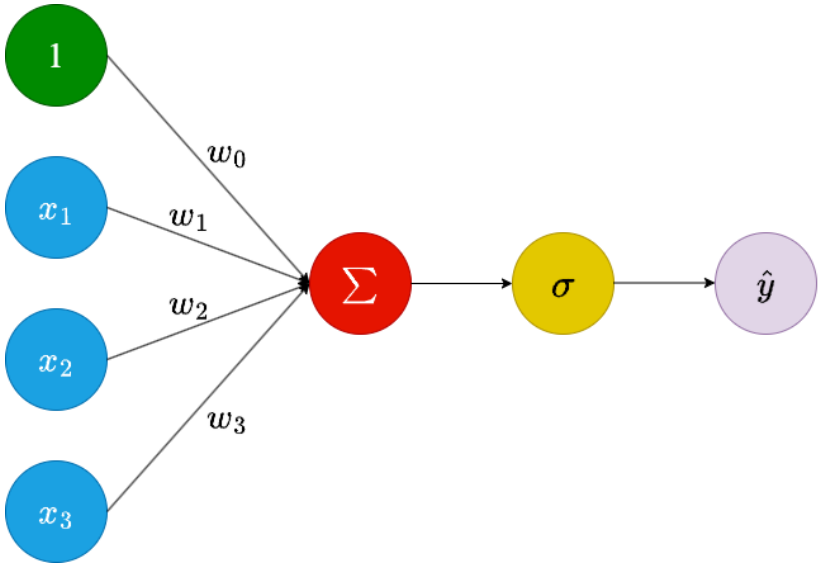


Figura 1: Estructura de un perceptrón

rando con más precisión la complejidad del conjunto de datos. Un ejemplo de función de activación es Rectified Linear Unit (ReLU) Fig. 2.

2.2. Feedforward Neural Networks

Una *feedforward neural network* (FFN) $\mathcal{N}$ es una pareja (G, Σ) donde $G = (V, A)$ es el grafo de la arquitectura básica y $\Sigma = \{\sigma_v : \mathbb{R} \to \mathbb{R} | v \in H\}$ es una familia de *funciones activación*. Consideremos $I = \{I_1, ..., I_d\}$, el conjunto de inputs, y $O = \{O_1, ..., O_{d'}\}$, el conjunto de outputs.

Una vez fijados los pesos w_a por cada $a \in A$ y sesgos s_v por cada $v \in V \backslash I$, el FFN define una función $n : \mathbb{R}^d \to \mathbb{R}^{d'}$. Considerando $I_j \in I$ para $j \in \{1, ..., d\}$ y $x \in \mathbb{R}$ definimos

$$z_{I_j}(x) := x_j$$

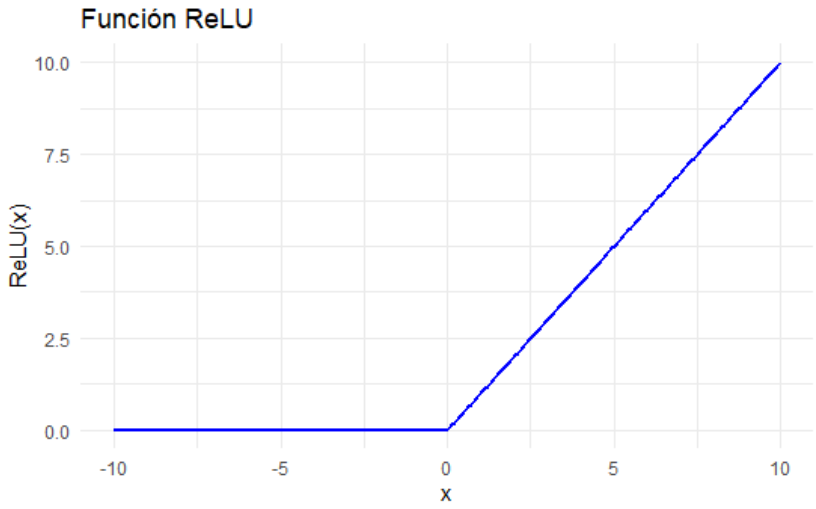


Figura 2: Función ReLU

La preactivación del vértice $v \in V \backslash I$ es:

$$z_v(x) := s_v + \sum_{e=(v',v)\in E} w_e \sigma_{v'}(z_{v'}(x))$$

La activación en el vértice v está dada por:

$$a_v(x) := \sigma_v\left(z_v(x)\right)$$

donde σ_v es la función de activación de v.

El output de la función n está dada por las preactivaciones en los vértices de output, es decir:

$$n(x) := \left(z_{O_1}(x), ..., z_{O_{d'}}(x)\right)$$

Antes de continuar con las estructuras siguientes, veamos cómo se entrenan los parámetros de una FFN. El método es el mismo para las estructuras más complejas. Nuestro objetivo es minimizar el error de la red neuronal, para eso tenemos que definir el error o pérdida. Definimos la función $\mathcal{L}(n(x^{(i)}; W), y^{(i)})$, donde $x^{(i)}$ es el conjunto de inputs, $y^{(i)}$, el conjunto de

datos reales que intentamos predecir (a partir del los inputs) y W el conjunto de pesos y sesgos de la red neuronal. Esta función mide la distancia de sus predicciones a los datos reales [AA24]. Sea $z^{(i)} = n(x^{(i)}; W) \in \mathbb{R}^n$ el output del modelo, una distribución de probabilidad (como en el caso de los transformers). La función estándar es *softmax cross-entropy loss* y se define como:

$$\mathcal{L}_{\text{softmax}}(z^{(i)}, y^{(i)}) = -\sum_{j=1}^{n} y_j^{(i)} \log \left(\frac{e^{z_j^{(i)}}}{\sum_{k=1}^{n} e^{z_k^{(i)}}} \right)$$

donde n es la longitud del vector $z^{(i)}$.

La medida que nos interesa es la suma de todas las perdidas del conjunto de datos de entrenamiento:

$$J(W) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{L}_{\text{softmax}}(z^{(i)}, y^{(i)}) = -\frac{1}{m} \sum_{i=1}^{m} \sum_{j=1}^{n} y_j^{(i)} \log \left(\frac{e^{z_j^{(i)}}}{\sum_{k=1}^{n} e^{z_k^{(i)}}} \right)$$

donde m es el número de ejemplos en el conjunto de datos de entrenamiento.

Por último, se calcula el gradiente $\frac{\partial J(W)}{\partial W}$ y se ajustan los pesos $W_{nuevo} = W - \eta \frac{\partial J(W)}{\partial W}$ para un $\eta > 0$ pequeño. Este proceso se itera hasta llegar a un mínimo local. Los pesos iniciales se suelen elegir aleatoriamente. Se conoce este método como *gradiente descente*.

En la práctica, hay modelos como PaLM de Google, que tiene 540 mil millones de parámetros. Naturalmente, las estrategias que se usan para calcular el gradiente se han ido sofisticando. Las más usadas en modelos basados en transformers son el gradiente descendente estocástico y el mini-batch.

2.2.1. El perceptrón multicapa completamente conectado

Un perceptrón multicapa (MLP) [AA24] es una red neuronal feedforward resultado de crear capas en secuencia de perceptrones apilados, de manera que el output de la capa anterior sirve de input para la posterior. En la Fig.

3 podemos ver la estructura simplificada. Las características de este tipo de FFN són:

- La arquitectura $G = (V, E)$ está estratificada en capas, es decir:

$$V = V^{(0)} \cup V^{(1)} \cup \cdots \cup V^{(L)},$$

  donde L es la profundidad de la red. Se tiene $I = V^{(0)}$ (inputs) y $O = V^{(L)}$ (outputs).

- Las aristas conectan únicamente capas consecutivas:

$$E \subseteq \bigcup_{l=0}^{L-1} V^{(l)} \times V^{(l+1)}.$$

- Todas las funciones de activación son iguales: $a_v = \sigma$ para todo $v \in H \cup O$.

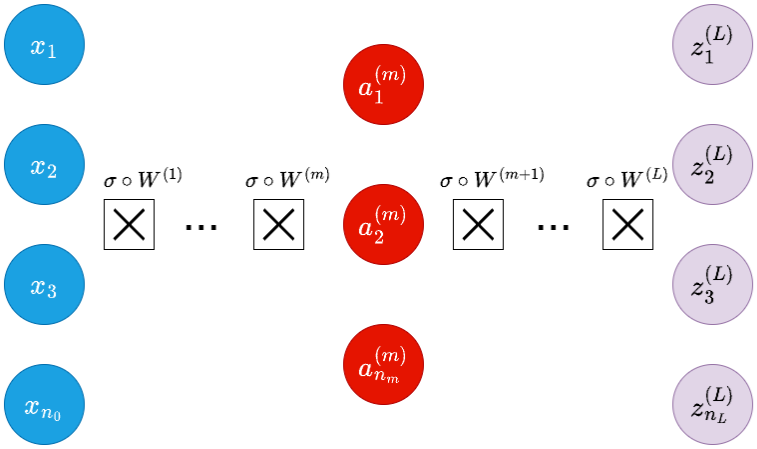


Figura 3: Estructura de un perceptrón multicapa completamente conectado

Decimos que el MLP es *completamente conectado* si:

$$E = \bigcup_{l=0}^{L-1} V^{(l)} \times V^{(l+1)}.$$

Las capas $V^{(1)}, \ldots, V^{(L-1)}$ se llaman *capas ocultas*, y denotamos $n_l := |V^{(l)}|$ como el ancho de la capa l-ésima. Sean $v_i^{(l)}$ los vértices de la capa l, $i = 1, \ldots, n_l$.

Fijado $\theta = (w, b)$ con los pesos y sesgos, definimos $b^{(l)} \in \mathbb{R}^{n_l}$ como el vector de sesgos en la capa l y $W^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ como la matriz de pesos cuyos elementos son:

$$W_{ij}^{(l)} := w_{(v_i^{(l)}, v_j^{(l-1)})}.$$

El modelo se describe recursivamente por capas. Dado $x \in \mathbb{R}^d$, definimos la preactivacion de la l-ésima capa como:

$$z^{(l)}(x) = b^{(l)} + W^{(l)} \cdot \sigma(z^{(l-1)}(x)), \quad \text{para } l = 0, \ldots, L$$

donde $z^{(0)}(x) := x$. Aquí σ actúa coordenada a coordenada.

Y definimos la activación de la l-ésima capa como:

$$a^{(l)}(x) = \sigma(z^{(l)}(x)), \quad \text{para } l = 0, \ldots, L-1$$

Por último, los outputs vienen dados por la preactivacion de la última capa, es decir, $n(x) = z^{(L)}(x)$.

Estas estructuras tienen su utilidad (y de hecho mas tarde veremos que finalmente forman parte de la arquitectura de los transformers, en los que se basan modelos como GPT) pero por si solas tienen varios problemas. Para empezar, no pueden captar la noción de tiempo o orden, todos los inputs son individuales y se procesan a la vez. Además, el tamaño del input es fijo, por lo que reduce mucho los casos en los que tiene utilidad. Si queremos un modelo que procese secuencias de datos, como lo es modelo de lenguaje, necesitamos otra arquitectura de redes neuronales. Con esto en mente, se

crearon los modelos secuenciales, como lo son los *recurrent neural networks* (RNNs) o long-short term memory (LSTM) [Ras21].

Se consiguen buenos resultados con estos modelos, pero siguen teniendo limitaciones que provienen de la recursividad del modelo. Usan un vector de memoria que tiene una dimensión determinada y, por tanto, tiene un limite de capacidad para retener información, lo que que puede dar problemas en frases largas. Son lentos, para analizar un vector de palabras se requiere el resultado del vector anterior. Estos problemas impulsaron un enfoque alternativo, buscar un mecanismo de atención que deduzca las dependencias globales entre input y output.

Las construcciones a continuación son las que forman parte de la arquitectura más avanzada hasta el momento, los *transformers*. Aunque los modelos de lenguaje actuales tienen ciertas peculiaridades, se basan en la arquitectura básica que veremos.

2.3. Tokenización, Embedding y Unembedding

Si queremos que nuestro modelo procese frases tenemos que encontrar primero la manera de representar palabras o trozos de la frase de manera numérica, lo que se conoce como codificar el lenguaje.

El transformer T toma como input una matriz $t \in \mathbb{R}^{n \times n_{\text{vocab}}}$, cuyas filas son vectores de la base canónica de $\mathbb{R}^{n_{\text{vocab}}}$. La parte llamada *decoder* del transformer opera sobre matrices $X \in \mathbb{R}^{n \times d}$ para algún $d < n_{\text{vocab}}$. Veremos el proceso de tokenización, que crea una función que envía una secuencia de palabras a una secuencia de tokens, que serán codificados a vectores y cómo funciona la aplicación $t \mapsto X$ [BK24].

Sea C una colección extensa de artículos, libros, blogs, mensajes que servirán para crear el *vocabulario* del transformer. Empezaremos considerando C una cadena larga de caracteres de un alfabeto estándar $\mathcal{A} = a, b, c, ..., A, B, C, ...$ Cada libro, artículo, etc. están separados en la cadena por un carácter especial.

El vocabulario $\mathcal{V}$ se construye mediante un proceso iterativo. El vocabulario inicial $\mathcal{V}_0$ es la codificación mediante ε_{UTF-8}, el proceso estándar para representar elementos de $\mathcal{A}$ como cadenas de bytes, de $\mathcal{V}$. Después, se escribe C usando exclusivamente el vocabulario inicial, creando la traducción C_0. Luego se itera el siguiente proceso:

- Se identifican los pares de caracteres (b, b') más frecuentes en C_i.
- Se crea la norma de fusión $\mathcal{F}_i$ que recuerda que b seguida de b' se escriben como un solo carácter bb'. $\mathcal{V}_i$ junto a la nueva norma de fusión forman $\mathcal{V}_{i+1}$.
- Se reemplazan todas las ocurrencias de (b, b') por bb' en C_i para crear C_{i+1}.

Esto continúa hasta que el vocabulario $\mathcal{V}$ alcanza el tamaño n_{vocab}. Por tanto, el vocabulario final $\mathcal{V} = (\mathcal{V}_0, (\mathcal{M}_i)_{i=0}^m)$, la unión del vocabulario inicial y todas las normas de fusión. A cada elemento de $\mathcal{V}$ se le llama *token*. Y tenemos una forma de representar una secuencia de palabras como una secuencia de tokens [BK24].

Fijado el vocabulario $\mathcal{V}$, se define una codificación one-hot:

$$\Omega : \mathcal{V} \longrightarrow \{e_j\}_{j=1}^{n_{\text{vocab}}} \subset \mathbb{R}^{n_{\text{vocab}}}$$

Así, cada cadena S se transforma en una secuencia:

$$(t_1(S), t_2(S), \ldots, t_N(S)) \subset \mathbb{R}^{n_{\text{vocab}}}$$

donde cada $t_i(S)$ es un vector one-hot (canónico) y $N = N(S)$ el numero de tokens de la secuencia.

El siguiente paso es empaquetar las secuencias en una longitud determinada, conocida como el tamaño de la *ventana de contexto*. La expresaremos como n_{ctx}. Ahora, a partir de una secuencia de palabras S, que cumple

$N(S) = n_{ctx}$, podemos formar una matriz $t(S) \in \mathbb{R}^{n_{ctx} \times n_{vocab}}$ que tiene como filas los one-hot tokens $(t_1(S), t_2(S), \ldots, t_{n_{ctx}}(S))$ [BK24].

Nuestro próximo objetivo es hacer un *embedding* de la codificación de one-hot vectors en $\mathbb{R}^d$ con $d < n_{vocab}$.

Para pasar al espacio de dimensión d, se introduce una *matriz de embedding*:

$$W_E : \mathbb{R}^{n_{\text{vocab}}} \longrightarrow \mathbb{R}^d$$

con parámetros entrenables. Entonces dada la matriz de one-hot tokens t:

$$t \xrightarrow{\text{Embedding}} X = tW_E^T = \begin{bmatrix} [\text{——} x_1 \text{——}] \\ \vdots \\ [\text{——} x_i \text{——}] \\ \vdots \\ [\text{——} x_n \text{——}] \end{bmatrix} \in \mathbb{R}^{n_{ctx} \times d}$$

También se define una matriz de unembedding:

$$W_U : \mathbb{R}^d \longrightarrow \mathbb{R}^{n_{\text{vocab}}}$$

y se aplica sobre las filas de X para obtener:

$$X \xrightarrow{\text{Unmbedding}} XW_U^T = \begin{bmatrix} [\text{———} \tau_1 \text{———}] \\ \vdots \\ [\text{———} \tau_i \text{———}] \\ \vdots \\ [\text{———} \tau_n \text{———}] \end{bmatrix} \in \mathbb{R}^{n_{ctx} \times n_{vocab}}$$

A veces se impone la condición $W_U = W_E^T$, lo que se denomina *embedding atado*.

2.4. Mecanismo de atención

Como el modelo no tiene recurrencia, para que pueda usar el orden de la secuencia, necesitamos añadir información acerca de la posición de los tokens. Para esto, usamos un *codificador posicional*. Su forma más simple es una matriz P de dimensión $n_{ctx} \times d$ con parámetros entrenables, que se suma a X en el paso del embedding $X = tW_E^T + P$ [BK24].

La clave del éxito de los transformers sobre las otras estructuras neuronales es principalmente debida a los mecanismos de atención, con los que los modelos .ªprenden"la importancia de cada palabra de la secuencia. La parte que se encarga de esto son las capas de auto-atención que describimos a continuación.

Empezamos definiendo tres matrices de parámetros entrenables W_q, W_k en $\mathbb{R}^{d \times d_k}$ y W_v en $\mathbb{R}^{d \times d_v}$. Estas matrices seran las que extraigan de nuestra matriz X, la *query*, la *key* y el *value*. Haciendo una analogía con una búsqueda en internet, la quary sería lo que escribimos en el buscador; la key, una descripción del contenido de las páginas webs; y el value, las características de la web a las que queremos prestar atención. Siguiendo el símil, queremos que nuestra query y key sean lo más parecidas posibles para que la búsqueda sea satisfactoria [AA24]. ¿Cómo se traduce esto matemáticamente?

Nuestra key $K = XW_k \in \mathbb{R}^{n_{ctx} \times d_k}$, la query $Q = XW_q \in \mathbb{R}^{n_{ctx} \times d_k}$ y el value $V = XW_v \in \mathbb{R}^{n \times d_v}$. Buscamos una forma de comparar la similitud de K y Q. Definimos entonces la proximidad de K y Q como $\frac{QK^T}{\sqrt{d_k}} \in \mathbb{R}^{n_{ctx} \times n_{ctx}}$, que nos da una matriz que describe la importancia de la relación cada componente dos a dos, por ejemplo, el valor de la primera entrada de la segunda fila es la relación del primer y segundo de token de la secuencia. Después, aplicamos la función activadora softmax sobre la matriz, que devuelve valores entre 0 y 1. Por último, extraemos las características que nos interesan con los pesos que hemos calculado antes (que nos indican a que dar más atención), multiplicando por la matriz V, de manera que obtenemos el output $A(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right) V \in \mathbb{R}^{n_{ctx} \times d_v}$. Esta construcción se conoce

como *attention heads* (Fig. 4) [VSP+23].

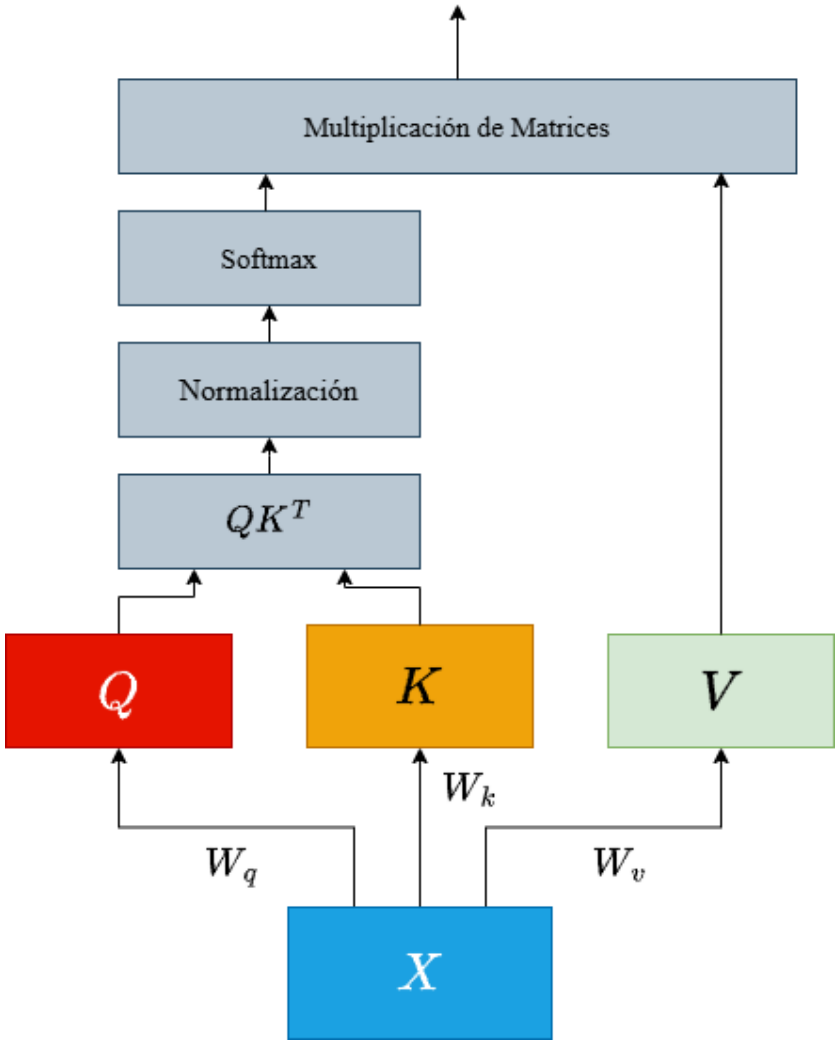


Figura 4: Esquema de una attention head

Los transformadores distribuyen h attention heads de manera paralela, para formar una *attention multi-head*. Cada attention head tiene un entrenamiento independiente de manera que ponen atención en características diferentes. Los h outputs se concatenan formando una matriz A_h de dimensión $n_{ctx} \times hd_v$ que vuelve a la dimensión deseada multiplicando por una

matriz de parámetros entrenados $W_O \in \mathbb{R}^{hd_v \times n_{ctx}}$. En la práctica, en vez de usar una query, key y value para cada attention head, entrenan proyecciones $W_i^Q, W_i^K \in \mathbb{R}^{n_{ctx} \times d_k}$ y $W_i^V \in \mathbb{R}^{n_{ctx} \times d_v}$ para cada attention head, que actúan sobre una única copia de $Q, K, V \in \mathbb{R}^{n_{ctx} \times n_{ctx}}$ para crear las query, key y value de esa attention head [VSP+23].

$$\mathrm{MH}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, \ldots, \mathrm{head}_h)\, W^O$$

$$\text{donde} \quad \mathrm{head}_i = \mathrm{A}(QW_i^Q,\, KW_i^K,\, VW_i^V)$$

2.5. Transformers

La arquitectura final de un transformer puede variar dependiendo de la finalidad del modelo que se está entrenando, pero, es una combinación de los elementos descritos. Un ejemplo simple de transformer es el siguiente.

Primero definimos dos estructuras formadas a partir de las anteriores.

Dado un attention multi-head MH y una MLP m, un *bloque residual* $B = B(MH, m) : \mathbb{R}^{n_{ctx} \times d} \to \mathbb{R}^{n_{ctx} \times d}$ se define como la composición de la capa de atención attn_{MH} con la capa feedforward ff_m [BK24], es decir:

$$B(MH, m) = \mathrm{ff}_m \circ \mathrm{attn}_{MH}$$

$$\mathrm{attn}_{MH}(X) := X + MH(X), \quad \mathrm{ff}_m(X) := X + m(X),$$

tenemos entonces:

$$B(X) = X + m\,(X + MH(X))$$

Sean $n, d \geq 1$ fijos, y sea n también el número de bloques en el transformador. Sea $\{MH_i\}_{i=1}^n$ una colección de attention multi-heads, todas con el mismo nuemro de attention heads $|MH_i| = n_{\text{heads}}$.

Sean también $\{m_i\}_{i=1}^n$ una colección de MLPs, cada una con L capas.

Entonces, el *decoder stack* D es la composición de los n bloques correspondientes [BK24]:

$$D : \mathbb{R}^{n\times d} \xrightarrow{B(HM_1,m_1)} \mathbb{R}^{n\times d} \xrightarrow{B(HM_2,m_2)} \cdots \xrightarrow{B(HM_n,m_n)} \mathbb{R}^{n\times d}$$

Ahora veamos el transformador completo.

Dada una matriz $t \in \mathbb{R}^{n_{ctx}\times n_{\text{vocab}}}$ de one-hot tokens, definimos el *transformador completo* T [BK24] como la composición de:

- El embedding (junto con el positional encoder): $t \mapsto tW_E^T + P$
- El decoder stack D
- El unembedding con W_U

Esto da lugar al siguiente diagrama de funciones:

$$T : \mathbb{R}^{n_{ctx}\times n_{\text{vocab}}} \xrightarrow{\text{Embedding}} \mathbb{R}^{n_{ctx}\times d} \xrightarrow{\text{Decoder}} \mathbb{R}^{n_{ctx}\times d} \xrightarrow{\text{Unembedding}} \mathbb{R}^{n_{ctx}\times n_{\text{vocab}}}$$

Como el input y el output del trasformer tiene la misma forma, podemos conectarlos en serie las veces que queramos. Por ejemplo, en el modelo de lenguaje GPT-1 el bloque esta repetido 12 veces [Pen22].

Por último, veremos brevemente como interpretar el output de un *modelo de lenguaje*. Un modelo de lenguaje no es más que una distribución de probabilidad. Dada una secuencia de palabras $S = (w_1, \ldots, w_{t-1})$, un modelo de lenguaje asigna la probabilidad de que cualquier palabra del vocabulario $\mathcal{V}$ continúe esa secuencia:

$$P(w_t \mid w_{1:(t-1)}), \quad \text{con } w_1, \ldots, w_{t-1}, w_t \in \mathcal{V}.$$

Con un modelo de este tipo se pueden generar nuevos textos: se comienza con una frase, luego se elige la palabra con mayor probabilidad (o se muestrea de acuerdo a la distribución) y se retroalimenta la secuencia extendida al modelo para predecir la siguiente palabra [Ras21].

Además, un modelo de lenguaje permite asignar una probabilidad total a una oración completa, utilizando la regla de la cadena de probabilidades condicionales:

$$P(w_{1:n}) = \prod_{i=1}^{n} P(w_i \mid w_{1:(i-1)}).$$

En un modelo de lenguaje, después de pasar la secuencia de palabras S (input) por los bloques transformadores obtenemos un output $T(t(S)) \in \mathbb{R}^{n_{\text{ctx}} \times n_{\text{vocab}}}$.

Sea $l(S) \in \mathbb{R}^{n_{\text{vocab}}}$ la última fila de la matriz $T(t(S))$. Llamamos a $l(S)$ los *logits* correspondientes a la predicción del próximo token dado el contexto S.

Para transformar $T(t(S))$ en una distribución de probabilidad, utilizamos la función *softmax* sobre vectores en $\mathbb{R}^{n_{\text{vocab}}}$, es decir: softmax : $\mathbb{R}^{n_{\text{vocab}}} \to \mathbb{R}^{n_{\text{vocab}}}$, donde, para $x \in \mathbb{R}^{n_{\text{vocab}}}$, la i-ésima entrada está dada por:

$$(\text{softmax}(x))_i = \frac{\exp(x_i)}{\sum_{j=1}^{n_{\text{vocab}}} \exp(x_j)}.$$

Conseguimos entonces, que todas las entradas del vector $(\text{softmax}(x))_i \in [0, 1]$, y que $\sum_{i=1}^{n_{\text{vocab}}} \text{softmax}(x)_i = 1$. Así, softmax$(x)$ define una distribución de probabilidad sobre el conjunto $\{1, \ldots, n_{\text{vocab}}\}$.

Definimos entonces:

$$P_S(i) := (\text{softmax}(l(S)))_i .$$

$P_S(i)$ es la probabilidad que el modelo asigna a que el siguiente token sea el i-ésimo en su vocabulario. En particular, $\arg\max_i P_S(i)$ se interpreta como el índice del token que más probabilidad tiene de ser el siguiente [BK24].

3. Aplicación de Large Language Models al Cap Set Problem

La primera motivación de este trabajo fue la aplicación de LLMs en un problema de matemática teórica, en concreto, un artículo de Nature [RPBN+23] en el que se mejoraba la cota inferior del cap set problem y se creaban nuevos algoritmos de bin packing mediante el uso de FunSearch; la combinación de un LLM congelado (sus parámetros se mantienen), llamado Codey, encargado de escribir el código que dé las soluciones deseadas, con un evaluador, que lo aleja de ideas incorrectas. Lo más novedoso es que, Codey, es un modelo preentrenado con código únicamente, es decir, sin ningún *fine-tuning* específico para los problemas que se tratan. La razón de esta decisión, aparentemente contraproducente, es buscar un enfoque más creativo al problema, sin el lastre de las soluciones convencionales. A continuación, veremos un resumen del método empleado y los resultados del artículo.

3.1. FunSearch

FunSearch es el sistema que utilizan para encontrar el código optimizado. La Fig. 5 muestra un esquema del funcionamiento del programa. Siguiendo el orden de la figura explicaré los componentes más detalladamente y como se adaptaron al caso concreto del cap set problem.

El primer paso es la especificación del problema, en cierta manera el prompt inicial. Proveen al programa con una función que evalúa las respuestas numéricamente y un primer programa que será el que evolucione. El programa tiene una parte que se mantiene, el esqueleto, y una parte que se irá modificando progresivamente, donde recae la parte critica que gobierna la lógica del programa. En el caso del capset, se evalúan los programas en función del tamaño del cap que generan, cuanto más grande sea, mejor. En la Fig. 6 podemos ver la especificación que usaron para iniciar la evolución del programa del cap set. La parte que evoluciona es la función priority, que

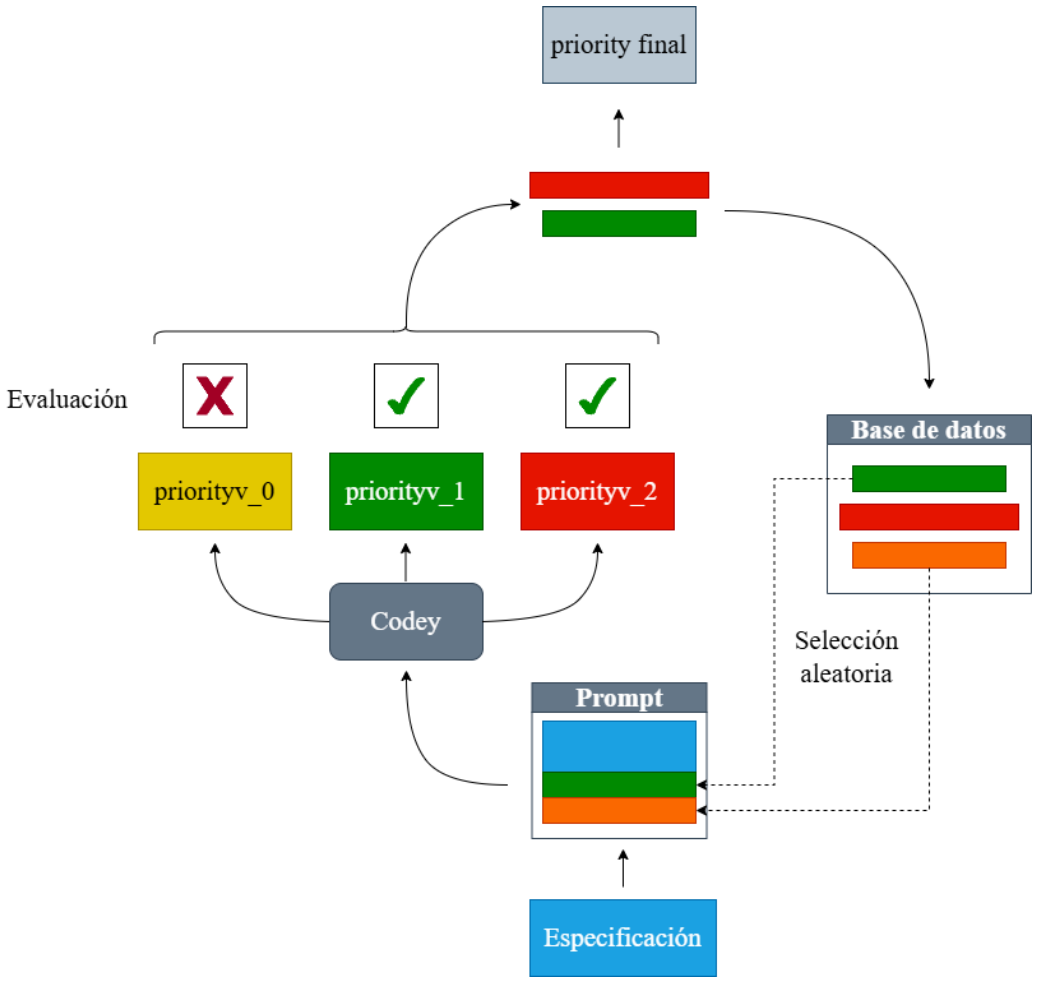


Figura 5: Esquema general de FunSearch

asigna a cada elemento de $\mathbb{F}_3^n$ un número, cuanto más grande, antes serán añadidos al cap set por la función solve (si son compatibles con los elementos dentro del cap set). El esqueleto en este ejemplo es la función solve, que añade elementos al capset en el orden de prioridad que nos da priority, bloqueando a la vez todos los elementos no compatibles; y evaluate, que devuelve el tamaño del cap set creado.

Este prompt inicial llega a Codey, el LLM preentrenado, y genera un

```python
"""Finds large cap sets."""
import itertools
import numpy as np


@funsearch.run
def evaluate(n: int) -> int:
  """Returns the size of an `n`-dimensional cap set."""
  capset = solve(n)
  return len(capset)


def solve(n: int) -> np.ndarray:
  """Returns a large cap set in `n` dimensions."""
  all_vectors = np.array(list(itertools.product((0, 1, 2), repeat=n)), dtype=np.int32)

  # Powers in decreasing order for compatibility with `itertools.product`, so
  # that the relationship `i = all_vectors[i] @ powers` holds for all `i`.
  powers = np.array([3 ** i for i in range(n - 1, -1, -1)], dtype=np.int32)

  # Precompute all priorities.
  priorities = np.array([priority(tuple(vector), n) for vector in all_vectors])

  # Build `capset` greedily, using priorities for prioritization.
  capset = np.empty(shape=(0, n), dtype=np.int32)
  while np.any(priorities != -np.inf):
    # Add a vector with maximum priority to `capset`, and set priorities of
    # invalidated vectors to `-inf`, so that they never get selected.
    max_index = np.argmax(priorities)
    vector = all_vectors[None, max_index]  # [1, n]
    blocking = np.einsum('cn,n->c', (- capset - vector) % 3, powers)  # [C]
    priorities[blocking] = -np.inf
    priorities[max_index] = -np.inf
    capset = np.concatenate([capset, vector], axis=0)

  return capset


@funsearch.evolve
def priority(el: tuple[int, ...], n: int) -> float:
  """Returns the priority with which we want to add `el` to the cap set."""
  return 0.0
```

Figura 6: Esqueleto de código usado para descubrir cap sets grandes

numero definido de programas. Para los dos casos tratados, decidieron usar un enfoque extensivo en cuanto a los programas generados, en oposición a un enfoque centrado en la calidad de los programas, y los resultados fureon obtenidos usando un numero total de muestras del orden de 10^6.

A continuación, evalúan cada programa generado por el LLM y les asignan una puntuación. Los programas que no cumplan con el estándar (no producen respuestas correctas o no se ejecutan en el tiempo o memoria establecidos) son descartados y el resto pasan a la base de datos del programa.

La base de datos del programa consiste en una colección de programas correctos con sus puntuaciones asignadas, que serán utilizados para crear un nuevo prompt para Codey. Con el objetivo de mantener la diversidad en la

muestra, utilizan un modelo de *islas* que evolucionan independientemente. Para crear el siguiente prompt, primero se escoge una de estas islas y luego el numero de programas que se requiera, favoreciendo a los que mejor puntuación tengan. Ocasionalmente, sustituyen los programas de las islas que peor desempeño tienen (con las puntuaciones más bajas en sus programas mejor valorados) por un clon de uno de los mejores programas de las otras islas.

Para crear el siguiente prompt se seleccionan aleatoriamente 2 programas de una isla, siguiendo el método descrito en el parágrafo anterior. Estos programas se combinan en un solo prompt, es decir, se sustituye la función priority por priority_v0 y priority_v1 en orden decreciente de puntuación. Se utiliza más de un programa para que el LLM pueda encontrar patrones y generalizarlos. Este prompt le vuelve a llegar a Codey, que inicia otra vez el proceso, repitiéndose tantas veces como se desee.

3.2. Resultados

Este método se diferencia de los SAT solvers porque busca en el espació completo, sin las restricciones añadidas que se usan para reducir el tiempo de ejecución. Además, no solo obtenemos un resultado, nos devuelve un programa capaz de construir el cap set, dándonos una noción de que es el conjunto generado. En materia de cap set, el avance conseguido fue en el de dimensión $n = 8$, se descubrió un cap set con 512 elementos (hasta el momento, el más grande conocido tenia 496 elementos).

También buscaron nuevos conjuntos admisibles de peso constante. Usaron una especificación con un esqueleto más complejo que en el cap set pero esencialmente son lo mismo: evoluciona una función priority y el resto simplemente comprueba que se cumplan las condiciones de ser admisible y evalúa en función del tamaño del conjunto. Examinando el código de la función priority, los investigadores notaron que se trataban las coordenadas de forma muy simétrica, y, de hecho, resulta que el conjunto admisible que construye

se conserva bajo permutaciones cíclicas independientes de las coordenadas dentro de cuatro grupos disjuntos de ternas de coordenadas [RPBN$^+$23]. Esta observación motivó formalización del concepto de conjuntos admisibles *simétricos* y un método para buscar estos conjuntos a partir de generadores, que llamaremos conjuntos *pre-admisibles*. A continuación veremos todas estas nociones.

Definición 3.1. *Llamamos conjunto admisible simétrico a un conjunto admisible $A \subseteq \mathbb{F}_3^n$, con $n = 3k$, si se preserva (como conjunto, es decir, reordenando sigue siendo el mismo conjunto) bajo todas las 3^k permutaciones cíclicas independientes de coordenadas dentro de los k grupos consecutivos disjuntos de ternas:*

$$[0,1,2],\ [3(k-1),3(k-1)+1,3(k-1)+2].$$

Definición 3.2. *Sea $S := \{0,1,\ldots,6\}$ y definamos una función de decodificación $\varphi : S \to \mathbb{F}_3^3$ como:*

$$\varphi(a) = \begin{cases} (0,0,0) & si\ a = 0 \\ (0,0,1) & si\ a = 1 \\ (0,0,2) & si\ a = 2 \\ (0,1,2) & si\ a = 3 \\ (0,2,1) & si\ a = 4 \\ (1,1,1) & si\ a = 5 \\ (2,2,2) & si\ a = 6 \end{cases}$$

Definición 3.3. *La órbita de $(x,y,z) \in \mathrm{Im}(\varphi)$ es el conjunto sin duplicados*

$$\mathcal{O}((x,y,z)) := \{(x,y,z),(y,z,x),(z,x,y)\},$$

es decir, la órbita bajo permutaciones cíclicas. Definimos la longitud de la

órbita $l : \mathrm{Im}(\varphi) \to \{1, 3\}$ *como:*

$$l(\varphi(a)) := \begin{cases} 1 & si\ a \in \{0, 5, 6\} \\ 3 & si\ a \in \{1, 2, 3, 4\} \end{cases}$$

Observación. *Los elementos decodificados* $\mathrm{Im}(\varphi)$ *son precisamente los elementos de* $\mathbb{F}_3^3$ *cuyas órbitas forman un conjunto admisible válido. Por ejemplo,* $(0, 0, 2)$ *genera el conjunto admisible* $\{(0, 0, 2), (0, 2, 0), (2, 0, 0)\}$*, mientras que* $(1, 1, 2) \notin \mathrm{Im}(\varphi)$ *genera* $\{(1, 1, 2), (1, 2, 1), (2, 1, 1)\}$*, que no es admisible.*

Definición 3.4. *El peso de un elemento* $a \in S = \{0, 1, \ldots, 6\}$ *se define como* $w : S \to \{0, 1, 2, 3\}$*, donde:*

$$w(a) = \#\{\textit{entradas no nulas en } \varphi(a)\} = \begin{cases} 0 & si\ a = 0 \\ 1 & si\ a \in \{1, 2\} \\ 2 & si\ a \in \{3, 4\} \\ 3 & si\ a \in \{5, 6\} \end{cases}$$

Definición 3.5. *Sea* $k \in \mathbb{N}$*. Un conjunto pre-admisible de dimensión* k *es un subconjunto* $P \subseteq \{0, 1, \ldots, 6\}^k$ *tal que:*

1. *Para cada par ordenado de vectores distintos* $(x, y) \in P^2$*, existe una columna* $1 \leq i \leq k$ *tal que* $w(x_i) < w(y_i)$*.*

2. *Para toda terna de vectores distintos* $x, y, z \in P$*, existe una columna*

$1 \leq i \leq k$ *tal que*

$$\begin{aligned}\{x_i, y_i, z_i\} \in \{&\{0,0,1\},\{0,0,2\},\{0,0,3\},\{0,0,4\},\{0,0,5\},\\ &\{0,1,2\},\{0,1,3\},\{0,1,4\},\{0,1,5\},\{0,1,6\},\\ &\{0,2,3\},\{0,2,4\},\{0,2,5\},\{0,2,6\},\{0,0,6\},\\ &\{0,3,4\},\{0,3,5\},\{0,3,6\},\{0,4,5\},\{0,4,6\},\\ &\{1,1,3\},\{1,1,4\},\{1,1,5\},\{1,1,6\},\{0,5,6\},\\ &\{1,2,5\},\{1,2,6\},\{1,3,4\},\{1,3,5\},\{1,3,6\},\\ &\{1,4,5\},\{1,4,6\},\{1,5,6\},\{2,2,3\},\{2,2,4\},\\ &\{2,2,5\},\{2,2,6\},\{2,3,4\},\{2,3,5\},\{2,3,6\},\\ &\{2,4,5\},\{2,4,6\},\{2,5,6\},\{3,3,5\},\{3,3,6\},\\ &\{3,5,6\},\{4,4,5\},\{4,4,6\},\{4,5,6\}\}\end{aligned}$$

Teorema 3.6. *Dado un conjunto pre-admisible P de dimensión k, el conjunto*

$$A := \bigcup_{x \in P} (\mathcal{O}(\varphi(x_1)) \times \cdots \times \mathcal{O}(\varphi(x_k)))$$

es un conjunto admisible simétrico en dimensión $n = 3k$, de tamaño:

$$|A| = \sum_{x \in P} \prod_{i=1}^{k} l(x_i).$$

Veamos ahora la razón de ser de las definiciones, una vez conocemos su finalidad, el teorema. Las primeras definiciones tienen el objetivo de facilitar la notación y la manipulación de los elementos. Es más fácil trabajar con vectores S^k que con vectores de ternas de puntos. El objetivo de la primera cualidad de los conjuntos pre-admisibles, la del peso w, es garantizar la primera condición de conjunto admisible para A. Si $(x, y) \in P^2$ y $w(x_i) < w(y_i)$, nos garantiza que hay una $j \in \{3i, 3i+1, 3i+2\}$ tal que $x_j = 0 \neq y_j$ (la condición (1) de admisible). La segunda condición de pre-admisible sirve para

garantizar que, dados $x, y, z \in A$ generados por elementos diferentes de P, existe una coordenada j tal que $\{x_j, y_j, z_j\} = \{\{0,0,1\},\{0,0,2\},\{0,1,2\}\}$, la condición (2) de admisible. Las 49 posibilidades son el resultado de hacer una comprobación sistemática de todas las combinaciones, de manera que, 3 elementos cualesquiera de las 3 órbitas cumplan la condición.

Esta observación y su posterior formalización, permitió adaptar FunSearch para buscar conjuntos pre-admisibles. Este método produjo los mejores resultados, ya que, al reducir el espacio donde buscaban, era posible aumentar las dimensiones y pesos. Descubrieron un conjunto admisible $I = (15, 10)$ *completo* (del máximo tamaño posible, $\binom{15}{10}$) y un conjunto admisible parcial en $A(24, 17)$ de 237984 elementos, que fue con el que obtuvieron la cota de $c < 2{,}2202$, la mejor hasta el momento.

Referencias

[AA24] Alexander Amini and Ava Amini. Conferencias en youtube, MIT Introduction to deep learning 6.S191. https://www.youtube.com/watch?v=alfdI7S6wCY, 2024.

[AD93] Noga Alon and Moshe Dubiner. Zero-sum sets of prescribed size. *Combinatorica*, 15(3):301–309, 1993.

[BB82] T.C Brown and J.P Buhler. A density version of a geometric ramsey theorem. *Journal of Combinatorial Theory, Series A*, 32(1):20–34, 1982.

[BK24] Spencer Becker-Kahn. Notes on the mathematical structure of GPT LLM Architectures. *arXiv:2410.19370*, 2024.

[CLP16] Ernie Croot, Vsevolod Lev, and Peter Pach. Progression-free sets in $\mathbb{Z}_4^n$ are exponentially small. *arXiv:1605.01506*, 2016.

[CLP24] Ernie Croot, Vsevolod F. Lev, and Péter Pál Pach. Past and future of the cap set problem. *arXiv:2408.02328*, 2024.

[Ede04] Yves Edel. Extensions of generalized product caps. *Des. Codes Cryptogr.*, 31(1):5–14, 2004.

[EG16] Jordan S. Ellenberg and Dion Gijswijt. On large subsets of $\mathbb{F}_q^n$ with no three-term arithmetic progression. *arXiv:1605.09223*, 2016.

[Gre04] Ben Green. Finite field models in additive combinatorics. *arXiv:math/0409420*, 2004.

[Gro18] Joshua Grochow. New applications of the polynomial method: The cap set conjecture and beyond. *Bulletin of the American Mathematical Society*, 56:1, 10 2018.

[Mes95] Roy Meshulam. On subsets of finite abelian groups with no 3-term arithmetic progressions. *Journal of Combinatorial Theory, Series A*, 71(1):168–172, 1995.

[Pen22] Carolin Penke. A mathematician's introduction to transformers and large language models. https://x-dev.pages.jsc.fz-juelich.de//2022/07/13/transformers-matmul.html, 2022.

[Ras21] Sebastian Raschka. Curso online de la Universidad de Wisconsin, intro to deep learning and generative models course. https://sebastianraschka.com/teaching/stat453-ss2021/, 2021.

[RPBN+23] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan Ellenberg, Pengming Wang,

Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 2023.

[Sze75] E. Szemerédi. On sets of integers containing k elements in arithmetic progression. *Acta Arithmetica*, 27(1):199–245, 1975.

[Tyr22] Fred Tyrrell. New lower bounds for cap sets. *arXiv:2209.10045*, 2022.

[VSP+23] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *arXiv:1706.03762*, 2023.