

Inteligencia Artificial Aplicada para la Economía

Profesores

Profesor Magistral

Camilo Vega Barbosa

Asistente de Docencia

Sergio Julian Zona Moreno



Arquitectura Transformer

La revolución de la IA generativa





IA Predictiva vs. IA Generativa: El cambio de paradigma



La inteligencia artificial ha evolucionado desde modelos predictivos, que analizan patrones conocidos para hacer predicciones específicas (como un meteorólogo anticipando el clima), hacia sistemas generativos capaces de crear contenido nuevo y original.

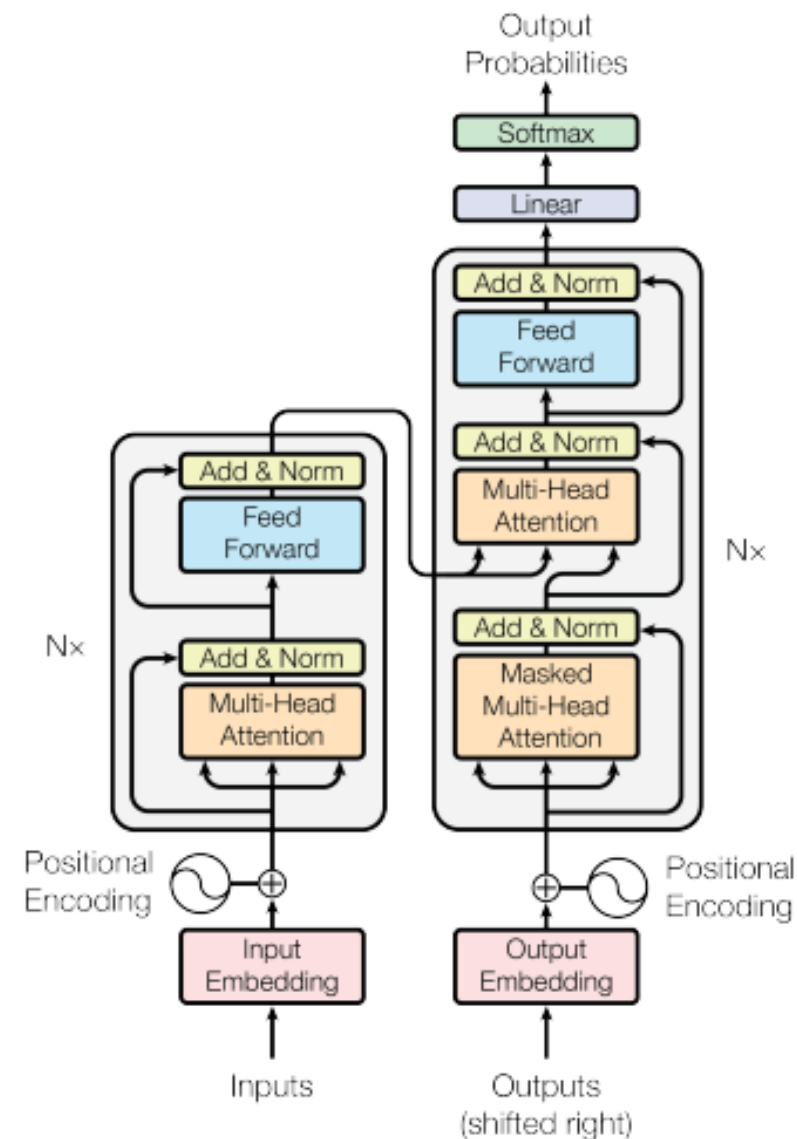


Esta transición representa un salto cualitativo en la capacidad de las máquinas: ya no solo pueden reconocer patrones y predecir resultados, sino que ahora pueden imaginar, crear y generar contenido original que nunca ha existido, similar a cómo un escritor compone historias basadas en su conocimiento previo.

La disrupción de los Transformers

La arquitectura Transformer, introducida en 2017 en el paper "**Attention is All You Need**" (arxiv.org/pdf/1706.03762), revolucionó la IA al permitir que los modelos comprendan profundamente el contexto y generen contenido coherente y creativo a escala sin precedentes.

A diferencia de sistemas anteriores, los Transformers pueden "ver" todas las palabras simultáneamente y establecer conexiones directas entre ellas, sin importar la distancia. Esta capacidad ha dado origen a tecnologías como ChatGPT, DALL-E y Midjourney, transformando radicalmente lo que la IA puede crear.





Transformer vs RNN: Una nueva forma de procesar secuencias



RNN (Arquitectura Previa)

- Procesamiento **secuencial** (palabra por palabra)
- Limitación de **memoria a largo plazo**
- **Lento entrenamiento** (no paralelizable)
- Dificultad con **contextos amplios**



Transformer

- Procesamiento **paralelo** (todas las palabras a la vez)
- **Conexión directa** entre cualquier par de palabras
- **Rápido entrenamiento** en GPUs
- Captura **relaciones complejas** entre palabras distantes (**Atención**)

El mecanismo de atención permite que cada palabra "preste atención" a todas las demás palabras simultáneamente, creando representaciones contextualizadas mucho más ricas.

Con Embeddings de posición sobran las RNN

Embeddings Posicionales

- A diferencia de las RNN, los Transformers **no procesan secuencialmente**
- Necesitan saber **el orden de las palabras**
- Los embeddings posicionales **codifican la posición** de cada token
- Utilizan funciones sinusoidales para mantener relaciones posicionales

"Attention Is All You Need"

- Título revolucionario que significaba: "no necesitamos RNNs"
- Reemplaza recurrencia con **conexiones directas** entre palabras
- **Elimina** el procesamiento secuencial obligatorio
- Permite **paralelización masiva** en hardware moderno

```
PE(pos, 2i) = sin(pos/10000^(2i/d_model))  
PE(pos, 2i+1) = cos(pos/10000^(2i/d_model))
```

Mecanismo de Atención: Conceptos básicos

La atención es como un "foco de luz" que permite al modelo concentrarse en partes relevantes de la entrada.

Funcionamiento Intuitivo

1. Para cada palabra, el modelo pregunta: "**¿A qué otras palabras debo prestar atención?**"
2. Calcula un **puntaje de relevancia** entre cada par de palabras
3. **Pondera** la importancia de cada conexión mediante softmax
4. **Combina** la información de todas las palabras según su relevancia

En la frase "El banco anunció tasas de interés", la atención permite que "tasas" se relacione fuertemente con "interés" y "banco", capturando su contexto financiero.

1 2
3 4

Mecanismo de Atención: Formalización Matemática



Componentes Matemáticos

- Query (Q): Lo que estamos buscando
- Key (K): Lo que podría coincidir con nuestra búsqueda
- Value (V): La información que queremos extraer



Fórmula de Atención

$$\text{Attention}(Q, K, V) = \text{softmax} \left(\frac{QK^T}{\sqrt{d_k}} \right) V$$

La atención calcula cuánto debe "mirar" cada token a todos los demás, luego combina la información de manera ponderada.

Ejemplo Mecanismo de Atención en GPT (1/5)

Matrices de Transformación Inicial

- **Matriz Q (Query):** Transforma los embeddings de entrada para crear vectores de consulta
 - $Q = XW^Q$ donde X son los embeddings de entrada
 - W^Q : matriz de parámetros entrenables que proyecta al espacio de consultas
- **Matriz K (Key):** Transforma los embeddings para crear vectores clave
 - $K = XW^K$
 - W^K : matriz de parámetros entrenables que proyecta al espacio de claves
- **Matriz V (Value):** Transforma los embeddings para crear vectores de valor
 - $V = XW^V$
 - W^V : matriz de parámetros entrenables que proyecta al espacio de valores

Ejemplo Mecanismo de Atención en GPT (2/5)

Cálculo de Puntajes de Atención

- Multiplicación Q y K:
 - Se multiplican los vectores Q y K (transpuestos): QK^T
 - Cada elemento (i, j) indica cuánto debe "prestar atención" el token i al token j
- Escalado para estabilidad:
 - Se divide por la raíz cuadrada de la dimensión: $\frac{QK^T}{\sqrt{d_k}}$
 - Esto evita valores extremos en el softmax cuando d_k es grande
- Aplicación de Softmax:
 - $\text{Weights} = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$
 - Convierte los puntajes en **pesos de atención** que suman 1

Ejemplo Mecanismo de Atención en GPT (3/5)

Vector de Atención Final

- Ponderación con los pesos:
 - Los pesos de atención se multiplican por los vectores V
 - $\text{Attention} = \text{Weights} \cdot V$
- Interpretación:
 - Cada token recibe información de otros tokens según su relevancia
 - Tokens con mayor peso de atención contribuyen más al vector final
 - Esto permite capturar dependencias a larga distancia

Ejemplo Mecanismo de Atención en GPT (4/5)

Proyección de Salida

- Matriz de proyección O :

La matriz O toma los vectores de atención y los proyecta al espacio original

$$\text{Output} = \text{Attention} \cdot W^O$$

W^O : matriz de parámetros entrenables que proyecta de vuelta al espacio del modelo

- Fórmula completa del mecanismo:

- $\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$

- $\text{Output} = \text{Attention}(Q, K, V) \cdot W^O$

Ejemplo Mecanismo de Atención en GPT (5/5)

Analogía Simplificada

- **Query (Q):** "¿Qué estoy buscando?"
 - Como buscar información específica en una biblioteca
- **Key (K):** "¿Qué información ofrezco?"
 - Como las etiquetas o índices de los libros
- **Value (V):** "¿Qué contenido apporto?"
 - El contenido real de cada libro
- **Atención:** El proceso de encontrar los libros más relevantes (K) para tu búsqueda (Q) y extraer su contenido valioso (V)

De Atención Simple a Multi-Head Attention

Atención Causal (Autoregresiva)

- Utilizada en **decodificadores** (como GPT)
- Cada token solo puede atender a tokens **previos**
- Permite **generación** paso a paso
- Evita "hacer trampa" viendo el futuro

```
X X X X X
  X X X X
    X X X
      X X
        X
```

Máscara causal: cada fila muestra a qué tokens puede atender cada posición

Multi-Head Attention

- **Múltiples** cabezas de atención en paralelo
- Cada cabeza aprende diferentes **tipos de relaciones**
- Una cabeza puede fijarse en **sintaxis**, otra en **semántica**
- Combina todas las representaciones para un **entendimiento más rico**

Es como tener varios "expertos" que analizan el texto desde diferentes perspectivas y luego combinan sus conocimientos.

Re-interpretaciones del Transformer

El Transformer original de 2017, aunque revolucionario, era **complejo y computacionalmente costoso** 🏗️ por su arquitectura de codificador-decodificador. Los investigadores pronto descubrieron que podían obtener resultados sorprendentes utilizando **solo una de sus mitades** ✂️, lo que dio origen a dos familias de modelos especializados.

Por un lado, modelos como **BERT** 🔍 aprovecharon exclusivamente el **codificador** para tareas de comprensión, empleando **atención bidireccional**. Por otro lado, **GPT** 🖋️ optó por usar solo el **decodificador** con atención unidireccional para destacar en generación. Esta **bifurcación evolutiva** 🌱 marcó el inicio de una carrera para determinar cuál de estos enfoques simplificados alcanzaría mayores capacidades.



De Transformer a GPT: El inicio de una nueva era

OpenAI y el Generative Pre-trained Transformer (2018)

OpenAI dio un paso transformador en 2018 al introducir GPT (Generative Pre-trained Transformer), una simplificación brillante del modelo Transformer original.

A diferencia del Transformer completo, GPT utilizó únicamente la parte del **decodificador**, descartando el codificador. Esta decisión de diseño resultó ser una revolución en el procesamiento del lenguaje natural.

El término "GPT" nace como acrónimo de "Generative Pre-trained Transformer", señalando el enfoque en la generación de texto mediante pre-entrenamiento a gran escala.



Radford et al. (2018) - ["Improving Language Understanding by Generative Pre-Training"](#)

🔑 La innovación clave de GPT-1

La principal contribución de GPT-1 fue su estrategia de entrenamiento en dos etapas:

1. **Pre-entrenamiento generativo:** El modelo aprende a predecir la siguiente palabra en textos sin etiquetar (BookCorpus, 7,000+ libros)
2. **Fine-tuning supervisado:** Adaptación del modelo pre-entrenado a tareas específicas con mínimos cambios arquitectónicos

Este enfoque permitió aprovechar enormes cantidades de texto sin necesidad de anotaciones costosas.



Arquitectura de GPT-1 - 12 capas de decodificador - 117M parámetros - Atención causal (autoregresiva) - Predice tokens futuros solo a partir de contexto previo

BERT: El enfoque alternativo solo-codificador

Google y la revolución bidireccional (2018)

Casi simultáneamente a GPT-1, Google presentó BERT con enfoque distinto:

- **Solo codificador** del Transformer original
- **Atención bidireccional** - puede procesar contexto en ambas direcciones
- Nuevas tareas de pre-entrenamiento:
 - **Masked LM**: Predecir palabras ocultas
 - **Next Sentence Prediction**: ¿Son consecutivas?

A diferencia de GPT (unidireccional), BERT "mira" en ambas direcciones, creando representaciones contextuales más ricas.

Aplicaciones principales: - Clasificación de texto - Extracción de información - Respuesta a preguntas

 [Devlin et al. \(2018\) - "BERT: Pre-training of Deep Bidirectional Transformers"](#)

🤔 GPT vs BERT: Enfoques complementarios

Dos caminos desde el Transformer original

💡 **GPT (Solo decodificador)** - Atención unidireccional (causal) - Generación de texto fluida -
Pre-entrenamiento: predecir siguiente token - Ideal para tareas generativas

🔍 **BERT (Solo codificador)** - Atención bidireccional - Comprensión profunda del contexto - Pre-
entrenamiento: predecir tokens ocultos - Ideal para clasificación y análisis

Cada enfoque tiene sus ventajas: BERT excede en comprensión pero carece de capacidad generativa natural, mientras que GPT genera texto fluido pero con visión contextual limitada.

Estos dos enfoques definieron familias de arquitecturas que evolucionaron en paralelo, cada una optimizada para diferentes tipos de tareas.

GPT-2: El salto hacia las capacidades emergentes

"Language Models are Unsupervised Multitask Learners" (2019)

Un año después, OpenAI sorprendió al mundo con GPT-2, un modelo que representó un salto cuántico en escala y capacidades:

- **Escalamiento masivo:** De 117M a 1.5B parámetros (10× mayor)
- **WebText:** Nuevo dataset curado de alta calidad (45M enlaces)
- **Zero-shot learning:** Capacidad para realizar tareas sin entrenamiento específico

La innovación más sorprendente fue el descubrimiento de que el simple escalamiento del modelo y los datos produjo "capacidades emergentes" - habilidades que nadie había programado explícitamente.

💡 Las habilidades emergentes del GPT-2

El poder inesperado del escalamiento

GPT-2 demostró capacidades que nadie había anticipado en un modelo sin entrenamiento específico:

- **Traducción inglés-francés** sin haber visto pares de traducción
- **Resúmenes coherentes** sin entrenamiento en resumen
- **Respuestas a preguntas factuales** sin una base de conocimiento explícita
- **Generación de texto** manteniendo estilo y contexto en párrafos largos

Descubrimiento fundamental

Los investigadores observaron que el modelo había adquirido capacidades que tradicionalmente requerían arquitecturas específicas y entrenamiento supervisado.

Como señalaron en el paper:

"Demostramos que los modelos de lenguaje comienzan a aprender estas tareas sin supervisión explícita."



Flujo GPT Teórico

Con un modelo Pre-Entrenado



Preprocesamiento en Modelos GPT

Del texto bruto a la representación numérica

El preprocesamiento transforma el texto humano en datos matemáticos que los Transformers pueden procesar. 🧠

Este proceso sigue una **secuencia esencial**: primero, el texto se **normaliza** 🖌️ preservando elementos clave; luego se divide en **tokens** ✂️ mediante algoritmos como BPE; estos tokens se convierten en **IDs numéricos**

1	2
3	4

 según el vocabulario del modelo; finalmente, se transforman en **vectores densos** 📊 (embeddings) y se les añade **información posicional** 📍 para mantener el orden de las palabras.

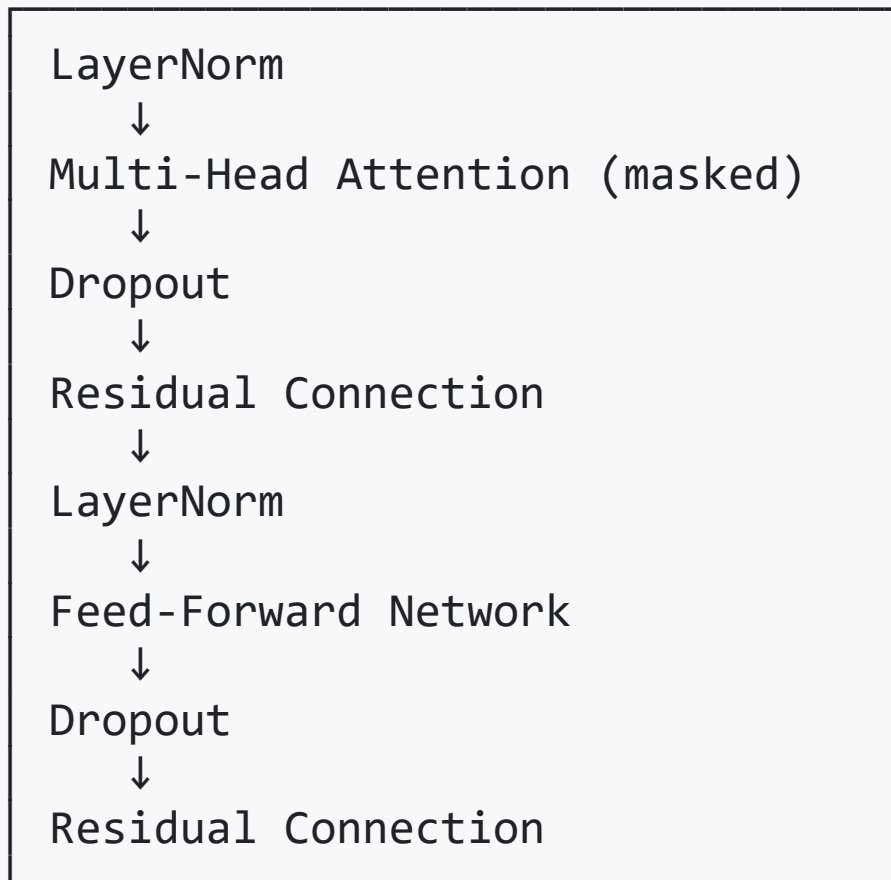
Flujo de Preprocesamiento: Un Ejemplo

De "Juan es muy..." a vectores

1. **Texto de entrada:** "Juan es muy..."
2. **Tokenización:** División en unidades léxicas ["Juan", "es", "muy"]
3. **IDs de tokens:** Conversión a identificadores numéricos [15873, 478, 1205]
4. **Embeddings de tokens:** Cada ID se convierte en un vector de alta dimensión
(La dimensión es determinante para la estructura del modelo, como # de parámetros)
5. **Embeddings posicionales:** Se generan vectores que codifican la posición de cada token
6. **Embeddings finales:** Suma de embeddings de token y posicionales
7. **Dropout de Embeddings:** Regularización temprana que desactiva aleatoriamente elementos del vector para prevenir sobreajuste, antes de entrar al primer bloque Transformer

🧩 Anatomía del Bloque GPT

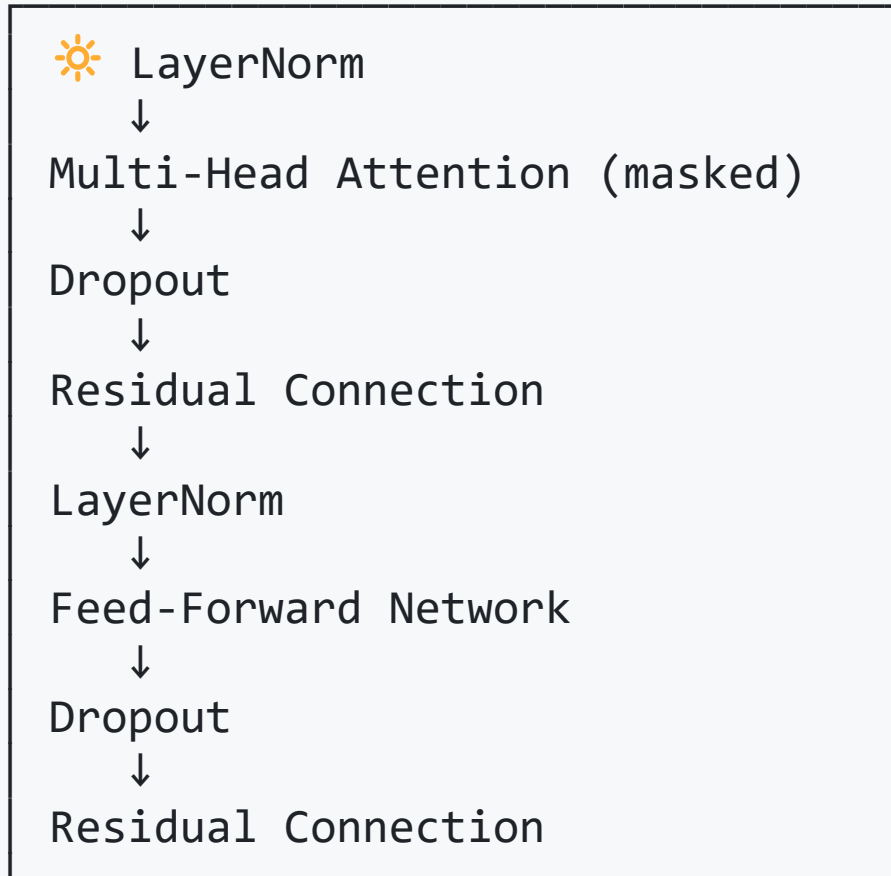
Descomponiendo cada componente clave



Cada bloque GPT es un sofisticado sistema de procesamiento de información que transforma gradualmente las representaciones de texto.

En las siguientes slides, exploraremos cada componente individualmente para entender su función específica.

LayerNorm: Estabilizando el aprendizaje

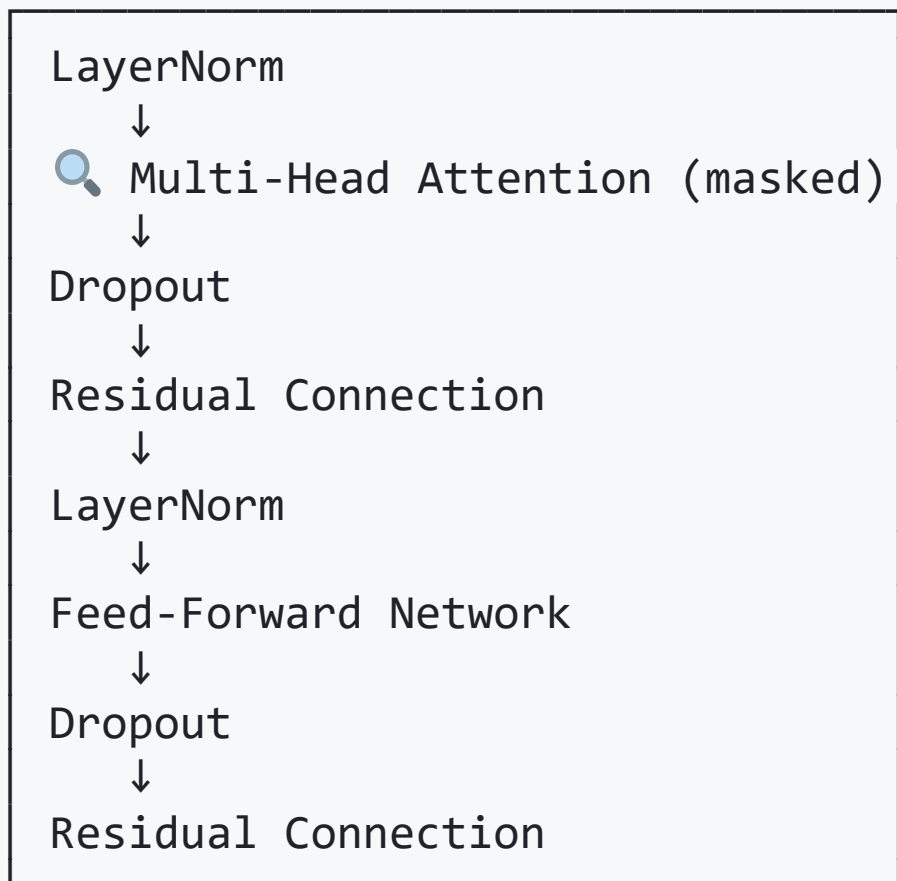


Layer Normalization

- **Normaliza** las activaciones para cada token individualmente
- **Estabiliza** el proceso de entrenamiento
- **Acelera** la convergencia de la red
- Aplica una **transformación afín** (γ , β) después de normalizar

A diferencia de BatchNorm, LayerNorm normaliza a través de las características de un solo token, no a través de múltiples ejemplos.

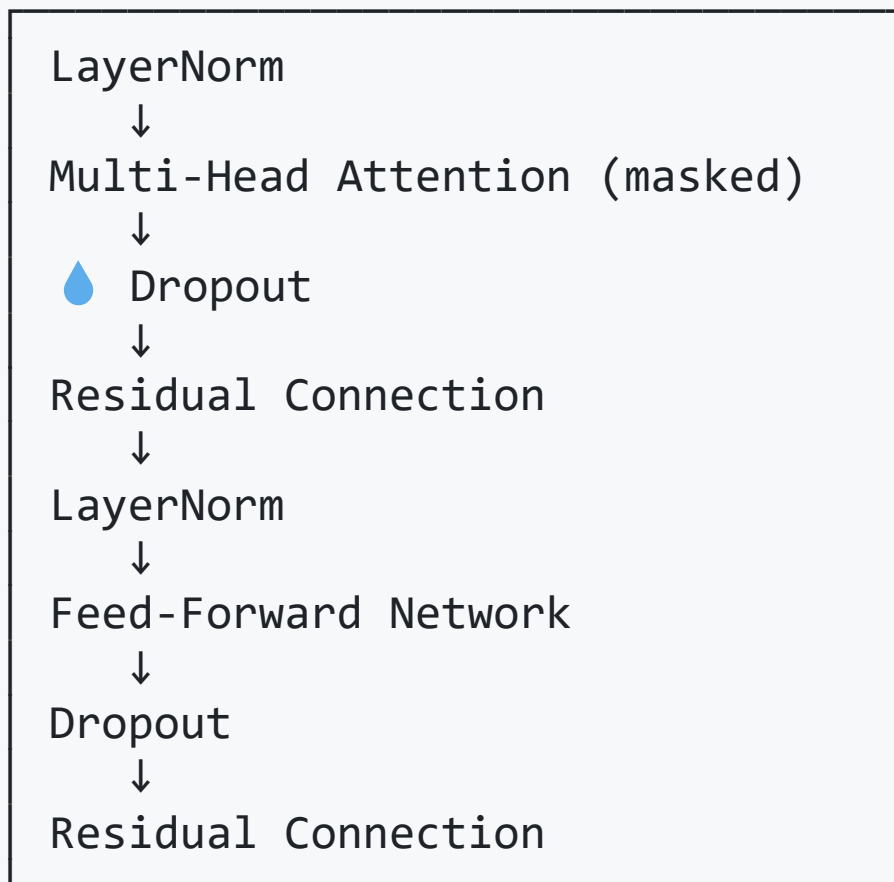
Multi-Head Attention: El corazón del Transformer



Multi-Head Attention (MHA)

- Permite al modelo **enfocar** diferentes aspectos del contexto simultáneamente
- Opera con **atención causal/enmascarada** (cada token solo ve tokens anteriores)
- Cada cabeza de atención aprende diferentes tipos de relaciones:
 - **Sintácticas**: estructura gramatical
 - **Semánticas**: significado y contexto
 - **Referencias**: conexiones entre entidades

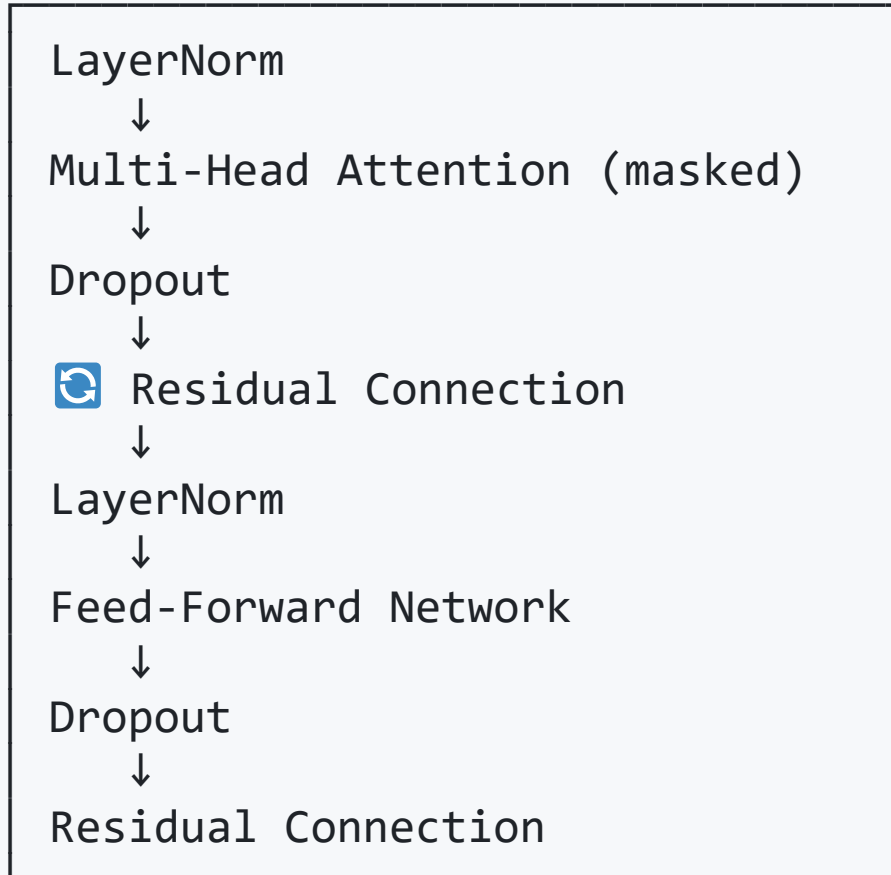
💧 Dropout: Regularización para prevenir sobreajuste



Regularización con Dropout

- **Desactiva aleatoriamente** un porcentaje de activaciones durante el entrenamiento
- **Previene el sobreajuste** forzando a la red a no depender de neuronas específicas
- **Mejora la generalización** del modelo en datos no vistos
- Típicamente con una tasa de 0.1 (10%) en GPT modernos

+ Residual Connection: Facilitando el flujo de información

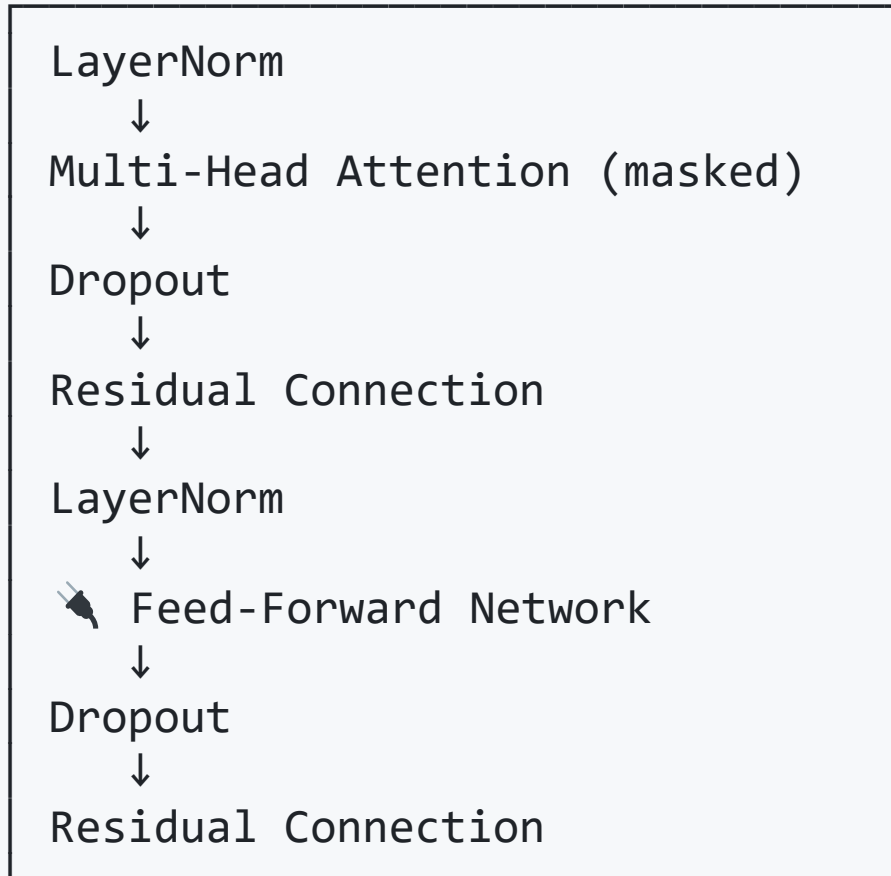


Conexiones Residuales

- **Suman** la entrada original a la salida de cada sublayer: $x' = F(x) + x$
- **Facilitan el flujo** de gradientes durante el entrenamiento
- **Mitigan** el problema de desvanecimiento de gradientes
- Permiten **entrenar redes más profundas** (12+ capas)
- Preservan la información original mientras se integran nuevas transformaciones



Feed-Forward Network: Procesamiento no lineal



Red Feed-Forward

- Procesa **independientemente** cada posición de token
- Consiste en **tres transformaciones**, dos lineales (expansión y contracción) y otra no lineal (GELU)
- Expande la dimensión interna (x4)
- La expansión + no linealidad permite **capturar patrones complejos** inaccesibles en el espacio original
- Comprime la información enriquecida de vuelta a la dimensión original del modelo



Flujo Completo del Bloque GPT

1. **Normalización inicial** (LayerNorm) de activaciones del token entrante
2. **Atención contextual** (Multi-Head Attention) para capturar relaciones entre tokens
3. **Regularización** (Dropout) para prevenir sobreajuste
4. **Conexión residual** para preservar información original
5. **Re-normalización** (LayerNorm) para estabilizar activaciones
6. **Procesamiento no lineal** (Feed-Forward Network) para transformar representaciones
7. **Regularización final** (Dropout) para prevenir sobreajuste
8. **Conexión residual final** para mantener flujo de información

Este ciclo se repite típicamente 12-96 veces en los modelos GPT modernos, refinando progresivamente las representaciones hasta capturar patrones lingüísticos extremadamente complejos.

Posprocesamiento en Modelos GPT

De vectores a texto generado

El posprocesamiento transforma las salidas matemáticas del modelo GPT en texto coherente que los humanos pueden leer. 🧠

Este proceso sigue una **secuencia fundamental**:

1. **Proyección Lineal** 📐: El modelo genera una matriz de vectores (logits) proyectando las representaciones internas a un espacio de dimensionalidad del vocabulario
2. **Distribución de Probabilidad** 🎲: Aplica softmax para convertir logits en probabilidades
3. **Muestreo** 🎯: Selecciona tokens basándose en estas probabilidades (greedy, temperatura, top-k, nucleus)
4. **Decodificación** 📝: Convierte los token IDs a texto legible y los concatena 🔄 con el contexto para continuar la generación

12 ¿Qué es la Proyección Lineal en GPT?

La **proyección lineal**, o capa de salida, transforma el output del bloque GPT en logits 

Output del bloque GPT (d=4):

```
[0.7, 0.4, 0.2, -0.5] # "muy"
```

↑ Dado que es un modelo autoregresivo, se toma el último vector de la secuencia "Juan es muy"

Matriz proyección (d×VocabSize):

```
[[0.1, 0.3, -0.2, 0.5, 0.4],  
 [-0.3, 0.2, 0.6, 0.1, -0.5],  
 [0.4, -0.1, 0.2, 0.3, 0.7],  
 [0.2, 0.4, 0.1, -0.2, 0.3]]
```

Multiplicación matricial:

```
Vector × Matriz proyección = Logits
```

Flujo de Posprocesamiento: Un Ejemplo

De "Juan es muy" a "Juan es muy feliz"

1. **Texto de entrada:** "Juan es muy..."
2. **Salida del bloque GPT + Proyección Lineal:** Matriz con vectores de logits de la última palabra/token ("muy") que sale del GPT
[-0.4929, ..., 6.1345, ..., -0.6093]
3. **Distribución de Probabilidad (softmax):** Transformación de logits a probabilidades
[0.0001, ..., 0.4213, ..., 0.0001]
4. **Muestreo:** El ID 1876 se selecciona con la mayor probabilidad (0.4213)
Usando estrategia greedy en este ejemplo
5. **Decodificación:** El ID 1876 corresponde al token "feliz"
6. **Texto actualizado:** "Juan es muy feliz"

Postprocesamiento Avanzado (I)

Estrategias de muestreo

- **Temperature scaling:** Controla aleatoriedad
 - Alta (> 1): Más creativo
 - Baja (< 1): Más determinista
- **Top-k sampling:** Limita a los k tokens más probables
- **Nucleus sampling (top-p):** Considera tokens hasta alcanzar probabilidad acumulada p

Estas técnicas permiten equilibrar creatividad, coherencia y diversidad en el texto generado.

Ejemplo de Top-p ($p=0.9$)

[("feliz", 0.4), ("contento", 0.3), ("alegre", 0.2)]

Solo estos tres tokens serían candidatos para la selección.

Postprocesamiento Avanzado (II)

Filtrado y seguridad

Filtrado de contenido

- Detección de texto ofensivo o peligroso
- Clasificadores de seguridad
- Listas de términos prohibidos

Corrección automática

- Ajuste de gramática y puntuación
- Normalización de formatos
- Eliminación de repeticiones

Normalización

Original: "El sensor mide 5.4cm y pesa 8gr"

Normalizado: "El sensor mide 5,4 centímetros y pesa 8 gramos"

Los sistemas en producción suelen combinar varios niveles de filtrado para garantizar respuestas seguras y apropiadas.

Postprocesamiento Avanzado (III)

Personalización contextual

Desambiguación

- Resolución de referencias ambiguas
- Clarificación de respuestas vagas
- Expansión de abreviaturas técnicas

Adaptación

- Ajuste de tono y formalidad
- Especialización por dominio
- Integración con bases de conocimiento

El postprocesamiento contextual adapta las respuestas al contexto específico de uso.

Aplicaciones

- Asistentes: Personalización por usuario
- Documentación: Precisión terminológica
- Educación: Ajuste al nivel del estudiante



Arquitectura GPT Completa: Varios bloques en cascada

Token Embeddings + Positional



Bloque Transformer 1



Bloque Transformer 2



... (12-96x)



Bloque Transformer N



LayerNorm Final



Proyección a vocabulario

Arquitectura en cascada

- **Embeddings de entrada:** Convierten tokens a vectores + información posicional
- **N bloques transformer:** Procesan información en paralelo y en profundidad
 - GPT-2: 12-48 bloques
 - GPT-3: 12-96 bloques
 - GPT-4: Arquitectura no revelada, posiblemente 120+ bloques
- **Capa de salida:** Proyecta representaciones finales a probabilidades sobre el vocabulario



Entrenamiento del GPT

Del flujo teórico a la realidad práctica

Ahora que entendemos cómo funciona el **flujo completo de GPT** 🔄, podemos apreciar que todo este ingenioso mecanismo depende fundamentalmente de sus **parámetros** ⚙️. Sin un entrenamiento adecuado, todas estas sofisticadas operaciones matemáticas producirían solo **texto aleatorio e incoherente** ❌. El mismo flujo que hemos estudiado, con arquitectura idéntica pero parámetros sin optimizar, sería completamente inútil.

El **proceso de entrenamiento** 🏆 es lo que realmente da vida al modelo, ajustando millones o billones de valores numéricos en sus **embeddings** 📊, **matrices de atención** 👁️, **redes neuronales** 🧠 y **proyecciones** 📈. Esta optimización transforma una estructura matemática vacía en un sistema capaz de comprender contextos, capturar significados y generar texto con sentido. El verdadero **valor del GPT** 💎 reside precisamente en este proceso de encontrar los parámetros óptimos.

Lógica del entrenamiento

El entrenamiento de un GPT es fundamentalmente simple: el modelo toma muestras del **corpus tokenizado** y aprende a **predecir el siguiente token** en secuencia.

El modelo lee "El cielo es de color" y debe predecir "azul". Al principio falla, pero con cada intento **ajusta sus parámetros** para reducir el **error** entre su predicción y la realidad.

Este ciclo se repite hasta recorrer **todo el corpus tokenizado** y se repite **N veces** durante el entrenamiento. Como sabemos exactamente qué token debería venir después (está en el texto original), podemos **calcular con precisión el error** y refinar los parámetros.


```
Texto → Tokens → Predicción → Error → Ajuste → Repetir  
[----- Múltiples épocas -----]
```

Así, **token por token**, el modelo evoluciona desde generar texto aleatorio hasta producir contenido coherente.   

Flujo de Entrenamiento (I)

Del texto crudo al corpus curado

Todo comienza con una colección masiva de textos: libros, artículos, código, conversaciones... Este **texto crudo** pasa por un proceso de limpieza y filtrado para convertirse en el **corpus**, la base del conocimiento del modelo.



- El **corpus curado** no solo es limpio y diverso, también contiene estructuras útiles como separadores de documentos (`<|endoftext|>`).
- Su función es **dual**:
 - Construir el **vocabulario de tokens**
 - Proporcionar los **patrones lingüísticos y conceptuales** que el modelo aprenderá 

La calidad y representatividad de este corpus determinan el alcance del modelo. Es la materia prima de su "inteligencia".

Flujo de Entrenamiento (II)

De palabras a números: tokenización y vectores

Para que una red neuronal procese lenguaje, debemos traducir texto en números:



- **Tokenización** : Se convierte el texto en unidades mínimas con significado (tokens)
 - "Hola mundo" → [15873, 1205]
- **Vectorización inicial** : Cada token recibe un vector aleatorio (embedding)
 - "Hola" → [0.32, -0.75, 0.18, 0.92, -0.44, ...]

Estos vectores al principio no significan nada. Son puntos flotando en el espacio. El entrenamiento les dará estructura y significado progresivamente.

Flujo de Entrenamiento (III)

Contexto, ventanas y predicción

Con los textos ya convertidos en secuencias de tokens, comienza el entrenamiento real:

- **Ventanas de contexto (Chunks/Max Length)** : El corpus tokenizado se divide en bloques de tamaño fijo (por ejemplo, 1024 tokens).
Cada ventana es como una "página" que el modelo lee de una vez.
- **Aprendizaje autoregresivo** :
El modelo recibe una ventana y aprende a predecir cada token a partir de los anteriores dentro del mismo chunk.
 - Ej: [Juan, es, muy] → objetivo: feliz

Este principio de "predecir el próximo token" es el núcleo de todo el entrenamiento.

Flujo de Entrenamiento (IV)

Hiperparámetros clave en el entrenamiento ⚙️




- **Learning Rate** 📈: Controla el tamaño de los pasos en la actualización de parámetros
 - Muy alta → Inestabilidad y divergencia
 - Muy baja → Progreso lento o estancamiento en mínimos locales
 - Típicamente se usa un schedule que reduce la tasa gradualmente
- **Batch Size** 📦: Número de ejemplos procesados antes de actualizar parámetros
 - Batches grandes → Gradientes más estables, mayor memoria requerida
 - Batches pequeños → Más ruido en el entrenamiento, menor memoria

Otros hiperparámetros importantes: weight decay (regularización global), dropout rate (regularización dentro de los bloques del flujo GPT), número de épocas (iteraciones completas del dataset) y warmup steps (fase inicial de estabilización).

Flujo de Entrenamiento (V)

Medición del error y ajuste del modelo

¿Cómo sabe el modelo si está aprendiendo? A través de un proceso matemático preciso:

- **Predicción:** El modelo estima una probabilidad para cada posible token siguiente.
- **Función de pérdida (entropía cruzada) **:
 - Penaliza las predicciones incorrectas.
 - Cuanto más lejos esté la predicción del token correcto, mayor el error.
- **Retropropagación **: El error se distribuye hacia atrás a través de todas las capas del modelo.
- **Actualización de parámetros **: Usando optimizadores como **Adam**, el modelo ajusta sus pesos para reducir futuros errores.

Flujo de Entrenamiento (VI)

Organización interna del conocimiento

Durante el entrenamiento, los vectores y parámetros dejan de ser aleatorios. Emergen patrones útiles:

- **Los embeddings se agrupan:** palabras similares se acercan en el espacio vectorial.
- **Las capas de atención** aprenden relaciones: sujeto-verbo, causa-efecto, contexto semántico...

El modelo empieza a construir una **representación estructurada del lenguaje**, como si cartografiara el significado palabra por palabra, concepto por concepto.

Resumen: El Ciclo Completo de Entrenamiento

1. Corpus

Recolección y limpieza de textos masivos.

2. Tokenización

Conversión del texto a vectores (embeddings).

3. Ventanas

División en bloques (ej. 1024 tokens) para predecir el siguiente.

4. Hiperparámetros

Ajuste de learning rate, batch size, etc.

5. Optimización

Cálculo del error y actualización de pesos.

6. Capacidades

Aprendizaje de patrones semánticos y sintácticos.

Especialización por Profundidad en GPT

Especialización de las capas

Los estudios han mostrado que las diferentes capas del modelo se especializan:

- **Capas bajas:** Capturan patrones de palabras y sintaxis básica
- **Capas medias:** Procesan estructuras gramaticales complejas
- **Capas altas:** Manejan relaciones semánticas y razonamiento abstracto

Progresión de capacidades

Capas inferiores → Capas intermedias → Capas superiores

Patrones léxicos básicos → Estructura sintáctica → Semántica y razonamiento

A medida que la información fluye a través de las capas, las representaciones se vuelven más abstractas y capaces de capturar relaciones más complejas entre conceptos.

Parámetros en Modelos GPT

¿Qué son y dónde se encuentran?

- Los **parámetros** son valores numéricos ajustables que definen el comportamiento del modelo
- Son los "pesos" que el modelo utiliza para procesar información
- Se almacenan como vectores y matrices de números flotantes
- Cada parámetro individual es un número que el modelo aprende durante el **entrenamiento**

Parámetros en Modelos GPT

Ejemplo con Vectores de Embedding

- Un **vector de embedding** representa palabras o tokens como una secuencia de números
- Cada posición en el vector captura una dimensión semántica diferente
- Ejemplo de un vector de embedding (dimensión $d=8$):

[0.32,	-0.75,	0.18,	0.92,	-0.44,	0.01,	-0.36,	0.67]
↑	↑	↑	↑	↑	↑	↑	↑
p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8

- Cada valor p_i es un parámetro individual

Parámetros en Modelos GPT

Inicialización de Parámetros

- Al crear un nuevo modelo, los parámetros se **inician aleatoriamente**
- Típicamente se usan distribuciones como:
 - Distribución normal (Gaussiana)
 - Inicialización Xavier/Glorot
 - Inicialización He
- Esta aleatoriedad inicial es crucial para que el modelo comience a aprender

Parámetros en Modelos GPT

¿Cómo se calcula el número total de parámetros?

El número total de los parametros depende de la estructura y anatomia del modelo: vocabulario, dimensión, ventana de contexto, capas, cabezas...

Parámetros Entrenables en Modelos GPT (1/4)

Componente	Parámetros	Dimensiones
Token Embeddings	Matriz de embedding	[vocab_size, d]
Position Embeddings	Matriz de embedding	[max_len, d]
Layer Norm 1	Escala γ	[d]
	Desplazamiento β	[d]

vocab_size = tamaño del vocabulario

d = dimensión del modelo

max_len = longitud máxima de secuencia

γ (gamma) = parámetro de escala para normalización

β (beta) = parámetro de desplazamiento para normalización

Parámetros Entrenables en Modelos GPT (2/4)

Componente	Parámetros	Dimensiones
Multi-Head Attention	Matrices Q	$[d, d_{\text{head}} \times h]$
	Matrices K	$[d, d_{\text{head}} \times h]$
	Matrices V	$[d, d_{\text{head}} \times h]$
	Matriz O	$[d, d]$
	Sesgos (opcional)	[varios]
Layer Norm 2	Escala γ	$[d]$
	Desplazamiento β	$[d]$
Ventana de contexto	max_len	Escalar (por ejemplo, 1024)

d_{head} = dimensión de cada cabeza de atención

h = número de cabezas de atención

max_len = los tokens que puede procesar el modelo a la vez (en entrenamiento e inferencia).

Parámetros Entrenables en Modelos GPT (3/4)

Componente	Parámetros	Dimensiones
Feed-Forward	Matriz FF1	$[d, d_{ff}]$
	Sesgo FF1	$[d_{ff}]$
	Matriz FF2	$[d_{ff}, d]$
	Sesgo FF2	$[d]$
Layer Norm Final	Escala γ	$[d]$
	Desplazamiento β	$[d]$
Proyección	Matriz	$[d, \text{vocab_size}]$
	Sesgo	$[\text{vocab_size}]$

d_{ff} = dimensión de la capa feed-forward (típicamente $4 \times d$)

Ejemplo Cálculo Total de Parámetros (GPT-2 Base) (4/4)

Componente	Fórmula	Número de Parámetros
Token Embeddings	$\text{vocab_size} \times d = 50.257 \times 768$	38.597.376
Position Embeddings	$\text{max_len} \times d = 1.024 \times 768$	786.432
Atención ($\times 12$ capas)	$12 \times (4 \times 768 \times 768)$	28.311.552
Feed-Forward ($\times 12$ capas)	$12 \times (768 \times 3.072 + 3.072 \times 768)$	56.623.104
Layer Norms	$25 \times 2 \times 768$	38.400
Matriz de Proyección	768×50.257	38.597.376
TOTAL		124.439.808 ~ 124M

Nota: Con weight tying: ~85,8M

GPT-2 Base (sin sesgos): vocab=50.257, d=768, capas=12, cabezas=12, dim_ff=4xd (3.072)



De GPT a ChatGPT: La revolución conversacional

Evolución hacia asistentes IA alineados

🔍 A pesar de sus impresionantes capacidades, los modelos GPT originales generaban texto **sin entender realmente las necesidades humanas**. En 2022, OpenAI dio un paso revolucionario al transformar GPT en ChatGPT, convirtiendo un modelo generativo en un **asistente conversacional útil y seguro**.

⚙️ Esta transformación fue posible gracias a dos técnicas fundamentales: el **Fine-tuning con Instrucciones Supervisadas (SFT)**, que enseñó al modelo a entender y seguir indicaciones específicas, y el **Aprendizaje por Refuerzo con Feedback Humano (RLHF)**, que alineó el modelo con valores y preferencias humanas mediante un sofisticado sistema de votación.


Datos de Entrenamiento de GPT-3


Composición del corpus (Raschka, "LLM from Scratch")

Fuente	Descripción	Proporción	Tokens
Common Crawl	Web crawl data	60%	410B
WebText2	Web crawl data	22%	19B
Books1 y Books2	Internet-based book corpus	16%	67B
Wikipedia	High-quality text	3%	3B

499 mil millones de tokens en total, con énfasis en la calidad y diversidad.

Instrucción Supervisada: Primer paso

 El **primer gran avance** en la evolución hacia ChatGPT fue el **fine-tuning con instrucciones supervisadas**. Los investigadores descubrieron que, aunque GPT podía generar texto coherente, necesitaba aprender explícitamente a ser útil y seguir instrucciones específicas.

 Para lograr esto, se crearon **miles de ejemplos** donde humanos expertos escribían tanto la instrucción como la respuesta ideal. El modelo aprendió a imitar estas respuestas de alta calidad, transformándose de un simple generador de texto a un asistente capaz de entender y responder solicitudes específicas de los usuarios.

Dataset de Fine-tuning con Instrucción

Ejemplo simplificado

EJEMPLOS DE FINE-TUNING	
INSTRUCCIÓN	Explica qué es la fotosíntesis a un niño de 8 años.
RESPUESTA DESEADA	La fotosíntesis es como la forma en que las plantas "comen". Las plantas usan la luz del sol, agua y aire para hacer su propia comida. Es como si tú pudieras hacer un sándwich usando solo luz, agua y aire. La planta usa hojas verdes como "paneles solares" para atrapar la luz.

RLHF: Alineando con preferencias humanas

🔍 Aunque el modelo con instrucciones mejoró considerablemente, aún producía ocasionalmente respuestas **poco útiles, incorrectas o potencialmente dañinas**. El verdadero salto cualitativo llegó con el **Aprendizaje por Refuerzo con Feedback Humano (RLHF)**, que permitió alinear el modelo con valores y preferencias humanas.

🏆 El proceso es ingenioso: se generan **múltiples respuestas alternativas** para cada prompt, luego **evaluadores humanos las ordenan** de mejor a peor. Con estos rankings, se entrena un **modelo de recompensa** que puede predecir qué respuestas serán preferidas por los humanos. Finalmente, usando técnicas de aprendizaje por refuerzo, el modelo se optimiza para maximizar esta recompensa, aprendiendo a generar respuestas cada vez más útiles, seguras y alineadas.



El Sistema de Votación en RLHF

Capturando preferencias humanas

Proceso de ranking

1. Para cada instrucción, generar **múltiples respuestas**
2. Anotadores humanos **ordenan las respuestas**
3. Estas **preferencias** crean datos de entrenamiento

Los anotadores evalúan según: - ****Utilidad****
- ****Veracidad**** - ****Seguridad**** -
****Imparcialidad****



Ejemplo de Dataset RLHF

SISTEMA DE VOTACIÓN RLHF	
PROMPT: ¿Cuáles son los beneficios de la meditación?	
RESPUESTA A (PEOR)	<p>La meditación tiene muchos beneficios para la salud.</p> <p>Ranking humano: 1/5 ★</p>
RESPUESTA B (INTERMEDIA)	<p>La meditación ayuda a reducir el estrés, mejorar la concentración y promover el bienestar general.</p> <p>Ranking humano: 3/5 ★★</p>
RESPUESTA C (MEJOR)	<p>La meditación ofrece numerosos beneficios:</p> <ul style="list-style-type: none">• Reducción del estrés y ansiedad• Mejor concentración y atención• Mejora del sueño <p>Ranking humano: 5/5 ★★★</p>



Modelo de Recompensa y PPO: El cerebro del RLHF



El **modelo de recompensa** funciona como un juez automatizado que ha aprendido los criterios de calidad humanos. Este modelo se entrena con **pares de respuestas comparadas** (A es mejor que B) hasta que puede predecir confiablemente qué respuestas preferirían los humanos sin necesidad de preguntarles cada vez. Es como enseñar a un crítico gastronómico a calificar platos basándose en miles de opiniones previas.



Con este juez entrenado, entra en juego el **Proximal Policy Optimization (PPO)**, un sofisticado algoritmo que ajusta gradualmente el modelo original. El PPO **balancea dos objetivos aparentemente contradictorios**: mejorar las respuestas según el modelo de recompensa mientras evita alejarse demasiado del modelo original, preservando así sus capacidades lingüísticas.

PPO en acción: Un ejemplo simplificado

💡 Imaginemos un escenario donde el modelo debe responder a la pregunta "¿Cómo funciona un motor de combustión?":

- 1 El **modelo SFT inicial** genera varias respuestas posibles, desde explicaciones técnicas hasta analogías simplificadas.
- 2 El **modelo de recompensa** predice que los humanos preferirían una explicación que use analogías cotidianas y visuales, con un nivel técnico moderado.
- 3 El **algoritmo PPO** ajusta sutilmente los parámetros del modelo para que sea **más probable** que genere explicaciones con estas características, sin perder su capacidad de ofrecer detalles técnicos cuando sea apropiado.
- 4 Con cada **iteración de entrenamiento**, el modelo se alinea más con las preferencias humanas, aprendiendo no solo a explicar motores de combustión, sino a comunicar todo tipo de conceptos de la manera que los humanos encuentran más útil y accesible.

Modelo de Recompensa y PPO

Automatizando la evaluación de calidad

Modelo de Recompensa

- Predice qué respuesta preferirían los humanos
- Asigna una **puntuación** a cada respuesta
- Actúa como un **proxy automatizado** del juicio humano

Optimización con PPO

- Ajusta incrementalmente el modelo
- Maximiza la **recompensa** predicha
- Evita **desviaciones** excesivas del modelo SFT

RLHF: El ciclo de mejora continua

Ciclo de mejora iterativa:

- 1 Modelo entrenado con instrucciones genera respuestas
- 2 Humanos evalúan y ordenan las respuestas por preferencia
- 3 Modelo de recompensa aprende de estas preferencias
- 4 PPO optimiza el modelo usando recompensas predecidas
- 5 Modelo mejorado vuelve al paso 1 con nuevos casos

Este ciclo repetitivo de generación, evaluación y ajuste es lo que ha permitido a los modelos como ChatGPT mejorar continuamente.




🌟 El Resultado: ChatGPT, un asistente verdaderamente útil

🌟 El resultado de todo este proceso de refinamiento es **ChatGPT**, un asistente de IA fundamentalmente diferente de sus predecesores. A diferencia de los modelos generativos tradicionales, ChatGPT no solo produce texto coherente, sino que **entiende el contexto**, sigue instrucciones con precisión y mantiene conversaciones significativas que realmente responden a las necesidades de los usuarios.

🔧 Las mejoras son evidentes en múltiples dimensiones: ChatGPT genera respuestas más **útiles y detalladas**, es significativamente más **seguro** al evitar contenido dañino, mantiene **coherencia** a lo largo de conversaciones extensas y, quizás lo más importante, tiene la capacidad de **reconocer sus propias limitaciones** cuando corresponde. Todo esto fue posible gracias al revolucionario enfoque descrito por OpenAI en su investigación "**Training language models to follow instructions with human feedback**" (Ouyang et al., 2022).

Material complementario

Herramientas y videos para entender los Transformers

- 🔍 Visualización interactiva de Transformer: bbycroft.net/llm
- 🎬 Explicación paso a paso: [YouTube - 3Blue1Brown](#)
- 📺 DotCsv - Transformers:
 - [Parte 1](#) 
 - [Parte 2](#) 
- 🖋️ Canal "AI by Hand": [Tom Yeh](#) 

Recursos del Curso

Plataformas y Enlaces Principales

GitHub del curso

 github.com/CamiloVga/IA_Aplicada

Asistente IA para el curso

 Google Notebook LLM