

Apuntes semana 13 - 22/05/2025

Pamela González López 2019390545

Abstract—Este documento resume los contenidos de la semana 13, centrados en autoencoders, quantization y avances recientes en inteligencia artificial. Se revisan los componentes y tipos de autoencoders, el proceso de quantization y sus variantes, así como eventos destacados como Google I/O y el lanzamiento de Claude 4. También se introducen técnicas para reducir errores en modelos mediante calibración y selección de rangos. Finalmente, se anuncia la transición hacia el estudio del aprendizaje no supervisado.

I. NOTICIAS DE LA SEMANA SOBRE IA

Noticias relevantes de la semana sobre inteligencia artificial:

A. Google I/O 2025

Se comentó sobre el evento Google I/O, en el que se presentaron diversas innovaciones:

- **Android XR:** Colaboración para crear gafas de realidad aumentada que permitan ver contenido como Netflix directamente desde los lentes.
- **Gemini Agents:** Ecosistema completo para la creación y gestión de agentes inteligentes coordinados por modelos LLM.
- **Plan Ultra:** Suscripción premium de \$150/mes con acceso a herramientas avanzadas de IA.
- **Veo 3:** Modelo mejorado con capacidades para generar imágenes y sonidos realistas, incluso simulando física natural.

B. Conferencia de Microsoft

Se destacó que Microsoft anunció la liberación del código de *Copilot* como código abierto, así como del *WSL* (Windows Subsystem for Linux). Estos anuncios refuerzan el compromiso de Microsoft con el desarrollo abierto y los agentes inteligentes.

También se mencionó que una de las áreas en tendencia con alto potencial profesional es el desarrollo de agentes, herramientas que se orquestan mediante un modelo LLM.

Para expandir esta explicación, se introdujo **LangChain**, una herramienta que permite conectar un LLM con diversas funciones. Básicamente, se pueden construir pequeñas cadenas o *pipelines*, donde se define el *prompt*, lo que debe hacer el modelo, si debe conectarse a alguna herramienta o agente, y todo esto se organiza automáticamente mediante dicha estructura.

C. Claude 4 de Anthropic

La clase también abordó el lanzamiento de **Claude 4** por Anthropic. Este modelo superó varios *benchmarks*:

- 72.5% en *Software Engineering Benchmark*
- 43.2% en *Terminal Bench*

Claude 4 destaca en tareas de codificación y representa una fuerte competencia para los modelos de OpenAI.

II. AUTOENCODERS

Repasso de la clase anterior:

A. Encoder

Es un conjunto de bloques convolucionales seguidos de módulos de *pooling*. Su función principal es extraer las características más relevantes de la imagen de entrada y comprimir la información. La expectativa es que el encoder aprenda información significativa mediante la reducción progresiva de dimensiones (*downsampling*).

La idea es que, al utilizar el autoencoder para reducir la dimensión de los datos, se obtenga una distribución específica y estructurada en el espacio latente.

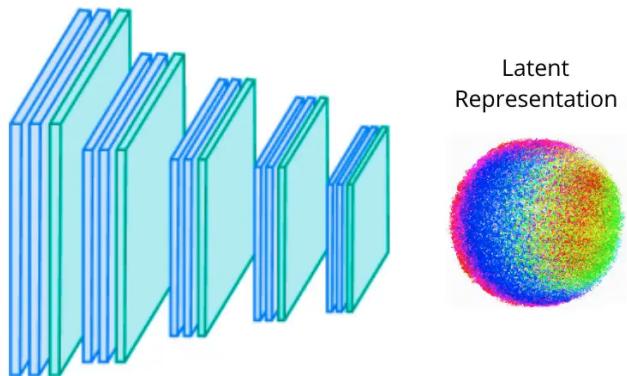


Fig. 1. Representación del cuello de botella

B. Cuello de botella

La parte intermedia, conocida como *cuello de botella*, concentra las características esenciales del dato. Es la parte más importante y pequeña del modelo. Es la representación en el espacio latente además restringe el flujo de información proveniente del encoder al decoder.

Es una zona estrecha en la red, donde en medio hay varias operaciones matemáticas (como capas convolucionales o totalmente conectadas, fully connected) que procesan la información.

El resultado de todo eso es un vector, que representa la información en el espacio latente. Ese vector es lo que el encoder logra entender de forma semántica sobre lo que hay en la imagen. ¿Y cómo se logra esa comprensión semántica? Gracias a las convoluciones que se aplicaron previamente en el encoder.

C. Visualización del espacio latente

Herramientas como *t-SNE* permiten visualizar la distribución de los datos en el espacio latente. En problemas simples, las clases tienden a agruparse de forma clara, lo que indica que el modelo ha realizado una buena codificación.

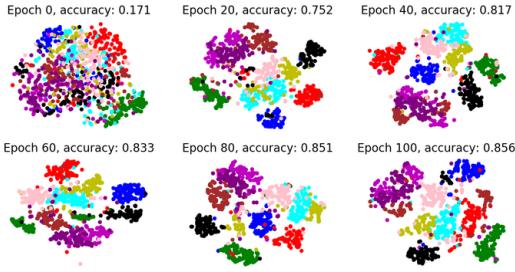


Fig. 2. Distribución en el espacio latente visualizada con t-SNE

Se puede ver cómo empieza a formarse una distribución en el espacio latente al analizar muchas imágenes, y cómo esa distribución va cambiando con pequeñas variaciones en las imágenes.

Se comenta que, si se tiene la oportunidad de aplicar t-SNE al proyecto, se verá una visualización en la que las 10 clases de audio procesadas estarán bien separadas. Cada clase aparecerá agrupada en su propio cluster, lo cual indica un buen desempeño del modelo. Esto se debe a que el problema planteado era relativamente sencillo, ya que las clases eran bastante distintas entre sí.

En cambio, si el reto hubiera consistido en distinguir entre voces femeninas y masculinas, o entre diferentes nacionalidades, la tarea habría sido más compleja. En ese caso, las diferencias entre clases serían más sutiles, debido a que las voces pueden compartir características similares en sus frecuencias. Sin embargo, para el problema actual, aplicar t-SNE permitirá visualizar claramente cómo el modelo logra separar las clases en el espacio latente.

D. Decoder

Es un conjunto de convoluciones que realizan *upsampling*, su función es reconstruir la imagen a partir del vector latente.

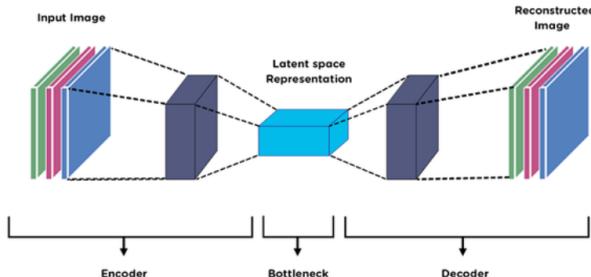


Fig. 3. Representación encoder y decoder

A partir de ese vector (la representación en el espacio latente), se aplican otras operaciones para reconstruir la

imagen original. Esta reconstrucción se realiza en el decoder, y el objetivo es que la imagen generada sea lo más parecida posible a la imagen de entrada.

E. Hiperparámetros a considerar

Se pueden experimentar diferentes configuraciones. Por ejemplo:

- ¿Cuántas capas utilizar en el encoder?
- ¿Cuántas capas usar en el decoder?
- ¿Aplicamos *skip connections* entre las capas del decoder?
- ¿Agregamos capas densas o solo conexiones directas entre bloques?

Existen múltiples variantes posibles, y cada configuración puede acercarnos a una arquitectura más efectiva. Este tema se destaca porque probablemente en el Proyecto 3 se trabajará con *autoencoders*, en particular con *Variational Autoencoders (VAE)*.

El objetivo será generar imágenes sintéticas, es decir, imágenes nuevas que no existen en el conjunto original. La idea es interpolar entre las representaciones del dataset, explorando otras posibles configuraciones en el espacio latente y generando así nuevos datos.

Tamaño del modelo y su impacto: El diseño del modelo depende del programador. Es importante tener en cuenta lo siguiente:

- Un modelo más grande será más complejo y podrá representar más información, pero también será más difícil de entrenar.
- Un modelo más pequeño será más rápido y fácil de entrenar, pero podría tener menor capacidad de representación.

Aquí es donde entra en juego el equilibrio entre **sesgo** y **varianza**: se debe encontrar un punto medio adecuado entre ambos extremos.

Evaluación del desempeño: Para evaluar qué tan bien se reconstruye la imagen, se utiliza el **Error Cuadrático Medio (MSE)**. Esta métrica permite medir qué tan parecida es la imagen reconstruida con respecto a la original.

F. Tipos de Autoencoders

- **Undercomplete Autoencoder:** Reduce la dimensionalidad para luego aumentarla y reconstruir la imagen.
- **Sparse Autoencoder:** Mantiene las dimensiones originales, pero durante el entrenamiento se apagan algunas neuronas para forzar al modelo a aprender una mejor reconstrucción. Existen distintas técnicas para implementar este "apagado".
- **Denoising Autoencoder:** Entrena al modelo para eliminar ruido en los datos. Esta técnica puede aplicarse incluso a tareas más complejas como *Super Resolution*.

G. Variational Autoencoders

Lo que se busca con el vector latente que genera el modelo es que siga una distribución normal (o una distribución conocida). ¿Por qué? Porque así podemos interpolar valores

dentro de esa distribución y generar nuevos datos que no estaban en el conjunto original.

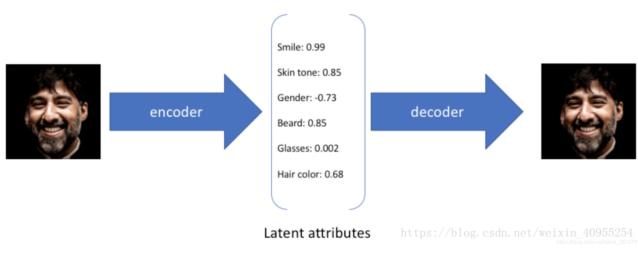


Fig. 4. Features

Por ejemplo, cuando tenemos un vector latente, cada una de sus dimensiones representa una característica (feature) semántica de la imagen, como:

- Qué tanto sonríe una persona
- El tono de piel
- La forma del rostro, etc.

Si esos features siguen una distribución de probabilidad, entonces podemos tomar cualquier punto dentro de esa distribución (aunque no lo hayamos visto antes) y generar una nueva imagen a partir de él.

Esto es precisamente lo que hace un Variational Autoencoder (VAE), aprende no solo a codificar las imágenes, sino también a generar nuevas combinaciones posibles en el espacio latente. Así puede crear imágenes nuevas que no ha visto durante el entrenamiento, pero que son coherentes con lo que aprendió.

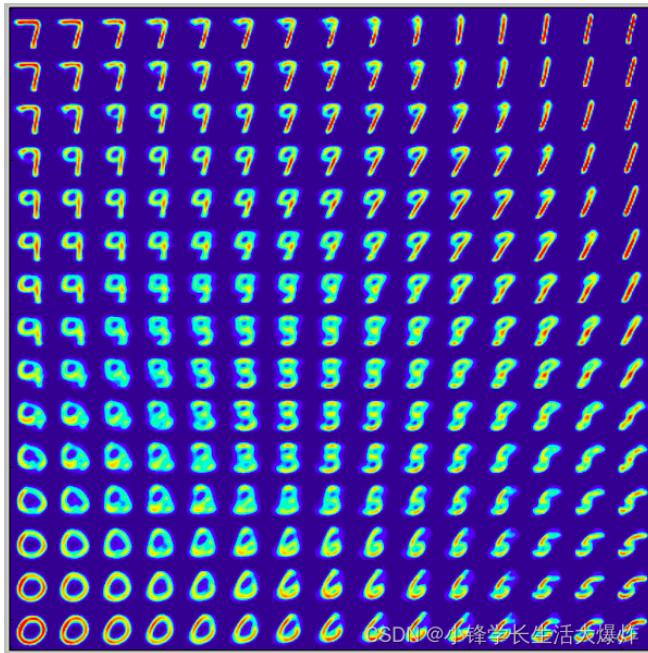


Fig. 5. Resultado autoencoder y conjunto de entrenamiento

Eso nos permite obtener resultados, algunas de las imágenes pueden ser del conjunto de entrenamiento, pero otras son generadas por el propio autoencoder.

Lo interesante es que existe una transición suave y continua entre una imagen y otra. No hay saltos bruscos ni cambios extraños. A medida que nos movemos a lo largo del vector latente (o de la distribución de probabilidad), el modelo va generando imágenes con pequeñas variaciones, pero que siguen siendo coherentes y comprensibles.

Esa es justamente una de las grandes ventajas del Variational Autoencoder.

Luego de eso el codificador produce dos vectores para hacer la distribución:

- La representación de μ
- La representación de σ (desviación estándar)

Reparametrización:

- Se toman muestras de μ y σ para producir el vector latente
- Se toma el vector latente y se alimenta al decoder.

Loss:

- Disminuir la suma de BCE (Binary Cross Entropy) y Kullback–Leibler Divergence (KLD)

H. Funciones de pérdida

- Error cuadrático medio (MSE): compara píxeles originales y reconstruidos.
- Kullback-Leibler Divergence (KL): mide la distancia entre la distribución generada y la distribución normal estándar.

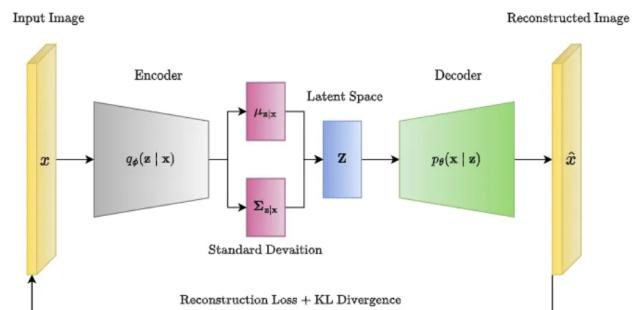


Fig. 6. Loss y KLD

Para evaluar qué tan bien se reconstruye una imagen, se utilizan dos elementos en la función de pérdida (loss):

1. Error Cuadrático Medio (MSE) Este se usa para comparar la imagen original con la imagen reconstruida. Se compara píxel por píxel: el valor de cada píxel original se compara con el del píxel reconstruido. La suma de esas diferencias nos indica qué tan bien se reconstruyó la imagen.

2. Kullback-Leibler Divergence (KLD) Esta mide qué tan diferente es la distribución aprendida (la que sale del codificador con media μ y desviación estándar σ) con respecto a una distribución normal estándar (media 0, desviación 1), que es la distribución objetivo.

En otras palabras:

- **La distribución aprendida** es la que el modelo trata de aprender.
- **La normal estándar** es la que queremos que siga.

El KLD se puede entender como una "distancia" entre esas dos distribuciones. Cuanto más pequeña es esta distancia, mejor se está ajustando el modelo a la forma deseada.

I. Ejemplo utilizado por el profesor

El profesor usó distribuciones de altura de dos poblaciones para explicar la divergencia KL, representándola como la diferencia entre un molde ideal (normal estándar) y la forma real de la distribución.

III. U-NET

De acuerdo con el paper *U-Net: Convolutional Networks for Biomedical Image Segmentation*

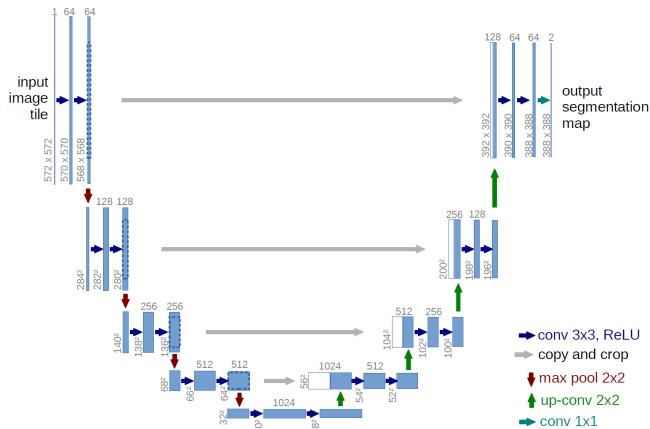


Fig. 7. Uso de skip connections en U-Net

U-Net es una arquitectura basada en autoencoders, pero incluye skip connections que conectan directamente las capas del encoder con las del decoder en el mismo nivel. Esto mejora la reconstrucción de imágenes.

Se menciona que la última tarea que tenemos pendiente es tratar de hacer segmentación de imágenes. Por ejemplo, si se tiene una foto de un perro y un gato, debe resaltar dónde está el gato y dónde está el perro en la misma imagen.

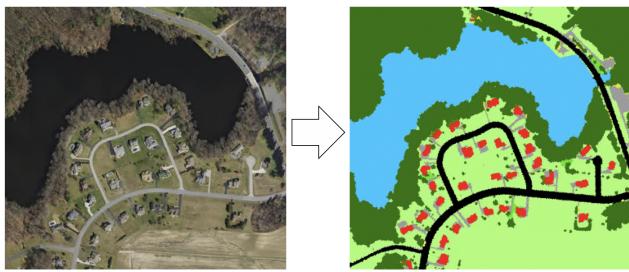


Fig. 8. Ejemplo de un segmentation map

IV. QUANTIZATION

Tema nuevo. Los modelos grandes, como LLaMA 2, pueden tener más de 70 mil millones de parámetros y requerir más de 28 GB de memoria solo para almacenarlos. Quantization reduce este tamaño al convertir los valores

flotantes en enteros. Haciendo el modelo mucho más pequeño.

Quantization no significa aplicar técnicas de redondeo a nuestros pesos y convertirlos a enteros. Es más complejo y necesita otros cálculos.

A. Ventajas del quantization

- Menor consumo de memoria: Los modelos ocupan menos espacio al cargarse.
- Menor tiempo de inferencia: Al usar datos simples, el modelo puede hacer predicciones más rápido.
- Menor consumo de energía: Al requerir menos operaciones de cómputo, la ejecución es más eficiente y barata.

B. ¿Cómo se representan los números en quantization?

Las computadoras usan un número fijo para representar los datos.

Podemos representar $2(\text{base})^N$ números distintos en una secuencia de bits

Binario	Número
000	0
001	1
010	2
011	3
100	4
101	5
110	6
111	7

Fig. 9. Representación del un número en binario

$$1 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 6$$

Los 1 son los bits encendidos el cero bit apagado. Así sería la representación de la casilla marcada en rojo 110.

Se representan los números en bloques enteros de 8 bits (byte).

- 16 bits / 2 bytes (short)
- 32 bits / 4 bytes (integer)
- 64 bits / 8 bytes (long)

En la mayoría de CPU los enteros están representados utilizando el complemento a 2.

- El primer bit indica el signo 0+, 1-
- El resto de bits indican el valor absoluto (si es positivo). El complemento (si es negativo).
- La ventaja es que se tiene una única representación del 0.

C. ¿Cómo se representan los puntos flotantes?

Incluyen números elevados al negativo de la base.

$$85,612 = 8 \times 10^1 + 5 \times 10^0 + 6 \times 10^{-1} + 1 \times 10^{-2} + 2 \times 10^{-3}$$

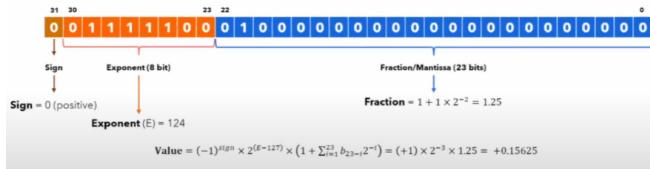


Fig. 10. Representación del número flotante

Primer bit, el 31 representa el signo, la parte naranja (30 al 23) sería el exponente que se usa para representar el número, y toda la parte azul (22 al 0) representa la fracción o mantisa.

Teniendo en cuenta esos datos, para representar el número se utiliza la fórmula, “value”.

D. Redes neuronales – Pesos

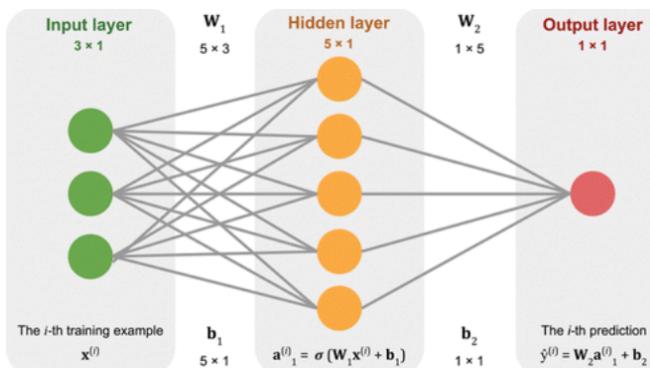


Fig. 11. Matrices de pesos

Las matrices de pesos son representadas con punto flotante. Se desea no perder calidad en los resultados cuando hagamos quantization.

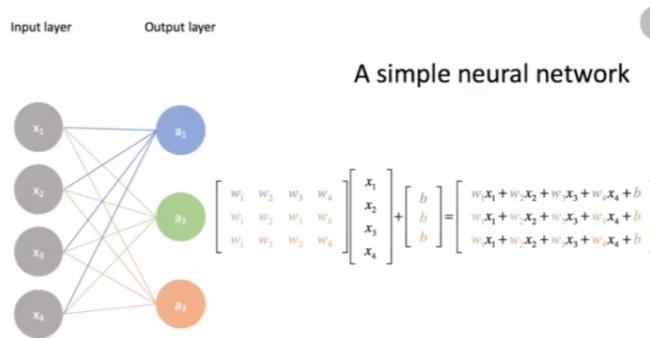


Fig. 12. Red neuronal

Todas esas operaciones se procesan utilizando aritmética de enteros.

Cuantizar:

- X

- W

- B

Descuantizar: Para procesar la siguiente capa

Queremos que la salida de las capas no cambie por aplicar quantization

Mantener el rendimiento del modelo, pero utilizando integers (beneficioso para la mayoría de los hardware especialmente embebidos).

Las capas siguientes no se deben de dar cuenta que las operaciones ejecutadas fueron hechas con enteros.

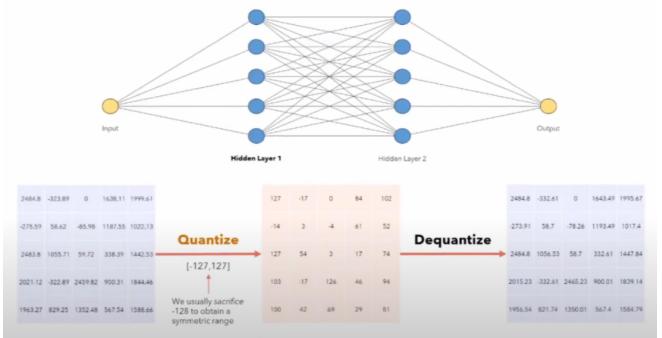


Fig. 13. Tablas de quantization y desquantization

Se ve que se pierde información al convertir de flotantes a enteros o de enteros a flotantes, pero se trata de que esa pérdida sea mínima.

E. Tipos de quantization

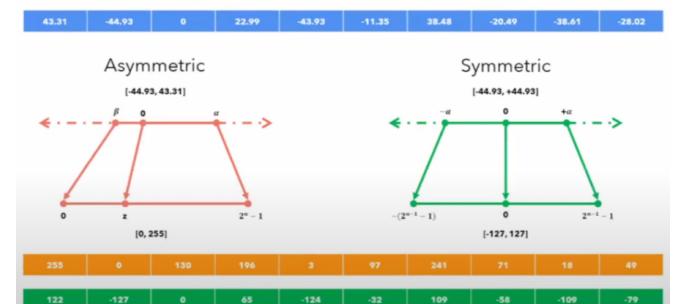


Fig. 14. Tipos de quantization

- **Simétrico:** valores entre -127 y 127, con cero centrado.
- **Asimétrico:** valores entre 0 y 255, utilizando un desplazamiento Z.

F. Quantization asimétrico

- Mapear valores entre un rango $[\beta, \alpha]$ a un rango $[0, 2^n - 1]$
 - β = el número menor de los valores a convertir.
 - α = el número mayor de los valores a convertir.

Fórmula:

$$x_q = \text{clamp}\left(\left\lfloor \frac{x_f}{s} \right\rfloor + z; 0; 2^n - 1\right)$$

- 1) x_f = Valor flotante
- 2) s : factor de parametrización, en cuánto se escala los datos.
- 3) z : desplazamiento con respecto al centro.
- 4) Aplicando la fórmula se acotan los valores entre 0 y $2^n - 1$.
- 5) n : número de bits.
- 6) Factor escalador: s

Parámetro de escalado s :

- $s = \frac{\alpha - \beta}{2^n - 1}$
- $2^n - 1$ = El rango de salida

Parámetro neutro z :

- $z = \left\lfloor -1 \cdot \frac{\beta}{s} \right\rfloor$
- n el número de bits

G. Desquantization asimétrico

- $x_f = s(x_q - z)$
- Permite volver al valor original
 - Introduce un grado de error, se hace un proceso inverso

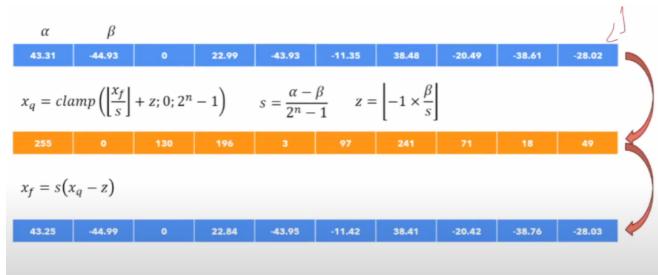


Fig. 15. Ejemplo asimétrico

H. Quantization simétrico

- Se mapean valores entre un rango en $[-\alpha, \alpha]$ a un rango $[-(2^n - 1), (2^n - 1)]$.
- Donde α = Mayor número absoluto.
- Usando 8 bits podemos representar un total de: $[-127, 127]$.

Fórmula

$$x_q = \text{clamp}\left(\left\lfloor \frac{x_f}{s} \right\rfloor; -(2^{n-1} - 1); 2^{n-1} - 1\right)$$

Parámetro de escalado s :

- $s = \frac{\text{abs}(\alpha)}{2^{n-1} - 1}$

I. Desquantization simétrico

Fórmula:

$$x_f = sx_q$$

Se debe multiplicar por el escalador, ya que no se utiliza un parámetro de neutro.

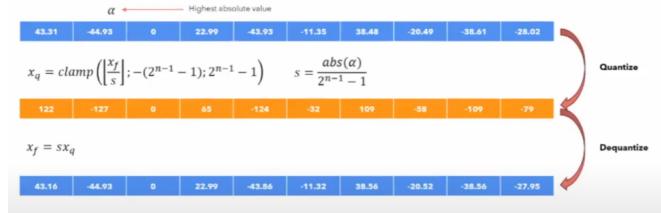


Fig. 16. Ejemplo simétrico

V. DYNAMIC QUANTIZATION

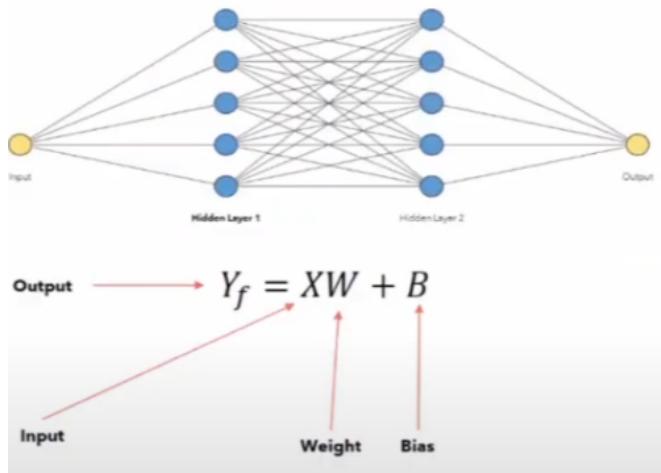


Fig. 17. Dynamic quantization

Transforma a enteros la entrada de la capa.
Queremos ejecutar todas las operaciones de enteros:

- W (Ya lo tenemos)
- B (Ya lo tenemos)
- X ¿Cómo cuantizamos la entrada x?

Las activaciones pueden ser cuantizadas en el momento, calculando el α y el β .

Una vez cuantizado el resultado Y_f será valores enteros. Para la siguiente capa se requiere los valores descuantizados (flotantes).

¿Cómo podemos calcularlos si no tenemos su parámetro escalador ni su cero? Se usa el proceso **Calibration**.

A. Calibration

- Se hace inferencia de las salidas usando algunas entradas y observando cuáles son las salidas típicas de las capas.
- Obtenemos un α y β razonables para calcular el escalador s y el valor z .
- Se realiza en la etapa *post-training quantization*.
- Transforma el Y cuantizado (que son enteros) a flotantes.

B. Estrategia de selección del rango

- En el caso asimétrico seleccionamos los valores de los extremos del tensor $[\beta, \alpha]$.

- En el caso simétrico seleccionamos el mayor valor en términos absolutos $[-\alpha, \alpha]$.
- No es la única estrategia de selección porque introducen más grado de error.

¿Cómo seleccionar $[\beta, \alpha]$?:

- Min – Max: $\alpha = \max(V)$, $\beta = \min(V)$. Es sensible a los outliers.

Si los valores del tensor estuvieran entre -50 y 50 y se presenta un outlier, este va a introducir un error mayor al momento de hacer la descuantización.

Original	43.31	-44.93	0	38.48	-20.49	1000.00	-28.02
Dequantized (Min-Max)	43.08	-45.08	0	24.59	-45.08	-12.29	36.88	-20.49	999.85	-28.48

Fig. 18. Outlier

C. Solución

Se puede utilizar la distribución de percentiles. En lugar de tomar los valores α y β como el máximo y el mínimo del conjunto, se seleccionan los valores correspondientes al percentil 99% y al percentil 1%. De esta manera, se busca acotar los outliers, reduciendo significativamente el error. Esta reducción del error debe analizarse excluyendo los outliers.

Original	43.31	-44.93	0	38.48	-20.49	1000.00	-28.02
Dequantized (Min-Max)	45.08	-45.08	0	24.59	-45.08	-12.29	36.88	-20.49	999.85	-28.48
Dequantized (Percentile)	43.38	-44.52	0	38.48	-20.49	50.00	-28.01

Fig. 19. Uso de percentiles

Errores son mucho más pequeños.

Otras estrategias

- **MSE:** Seleccionar un $[\beta, \alpha]$ de manera que el error de MSE entre el vector original y el vector cuantizado sea minimizado. *GridSearch* para encontrar los β, α .
- **Cross-Entropy:** Se usa cuando los valores en el tensor no son igual de importantes: (Softmax Layer LLM, Greedy, Top-P).

D. Quantization Granularity (Convolución)

- Las convoluciones están hechas por muchos filtros. Están aprendidos con valores y distribuciones distintas para cada uno. Observan distintos *features* de la imagen.
- No se puede aplicar el mismo valor β, α para todos los filtros. Mejor se busca valores de β, α para cada filtro.
- Se aplica convolución por canal.

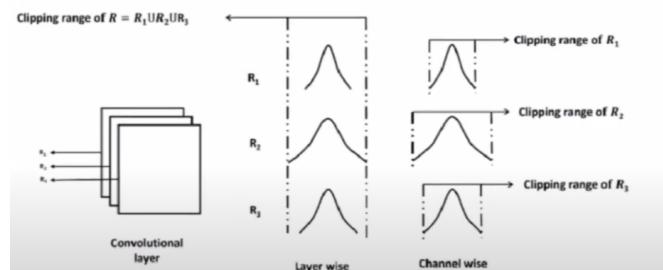


Fig. 20. Quantization Granularity

VI. PRÓXIMOS TEMAS

Al finalizar la clase, se anuncia que en la siguiente sesión se completará el tema de *post-training quantization* y se iniciará el estudio del **aprendizaje no supervisado**.