

Apuntes Semana 13 - 22/05/2025

Elaborado por: Joselyn Montero Rodríguez - c.2022136356

Abstract—Este documento presenta los apuntes del jueves 22 de mayo de 2025 de la clase de inteligencia artificial, cubriendo la estructura de autoencoders (encoder, cuello de botella, decoder), sus tipos (undercomplete, sparse, denoising, variational, U-Net) y consideraciones de diseño. También se explica la cuantización en redes neuronales, métodos simétrico y asimétrico, ventajas, calibración y selección de rango. Además se incluyen noticias relevantes sobre avances en inteligencia artificial presentadas durante la semana.

I. NOTICIAS DE LA SEMANA

- Android XR: Plataforma de Google diseñada para dispositivos de realidad aumentada que integra la inteligencia artificial para ofrecer experiencias inmersivas. Enlace a la noticia sobre Android XR.
- En el evento de Google I/O 2025 se presentó Gemini 2.5 Pro, el cual es un modelo con su propio agente de pensamiento.
- Google DeepMind presenta Veo 3, su modelo de generación de video más avanzado, permite crear videos de alta calidad en 4K con mayor control, consistencia y creatividad, siguiendo prompts de texto o imágenes. Enlace a la página Veo 3.
- Microsoft anunció que Visual Studio Code se convertirá en un editor de código de inteligencia artificial de código abierto al liberar el código de la extensión GitHub Copilot Chat bajo la licencia MIT. Enlace a noticia sobre Copilot como código abierto.
- Se habló de que se presentaron muchas noticias relacionadas a agentes y todo el ecosistema para construirlos, por lo que se recalca la tendencia del desarrollo de agentes que hay en este momento. Se hace mención de la plataforma LangChain, esta conecta modelos de lenguaje grandes (LLMs) para crear aplicaciones como chatbots o sistemas de generación de contenido, organizando agentes en una cadena para realizar tareas. Enlace a la página LangChain.
- Anthropic lanzó Claude 4 Opus y Claude Sonnet 4, modelos de IA avanzados que destacan en codificación, razonamiento complejo y tareas autónomas de larga duración. Enlace a la noticia lanzamiento de Claude 4.

II. REPASO CLASE DEL MARTES

A. Partes del autoencoder

- 1) Encoder: busca hacer una reducción de nuestro dato en un vector. Este vector al final se vuelve una representación más pequeña de nuestra información inicial. Al final la idea es obtener el espacio latente.
- 2) Cuello de botella: es la representación en el espacio latente, la reducción del autoencoder nos lleva ahí, en

medio se le pueden aplicar diferentes operaciones, lo importante es que al final deja un vector.

En la figura 1 se puede observar el cómo se ve la distribución de datos conforme se van haciendo las convoluciones. Se recomienda hacer la prueba con t-sne con el proyecto 1 para ver el comportamiento de las clases.

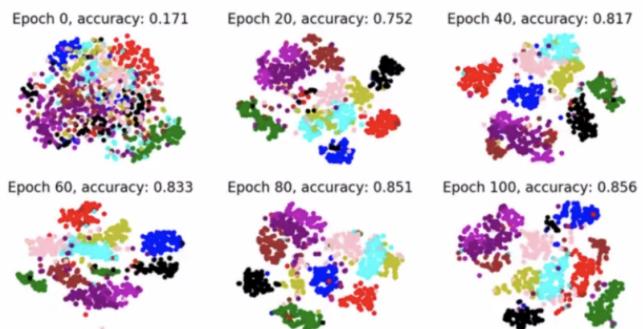


Fig. 1. Ejemplo gráficas de distribución de datos

- 3) Decoder: La convolución es un decoder, la idea después de esto es con otras operaciones reconstruir la imagen basada en el vector latente.

En la figura 2 se puede observar la estructura general de un autoencoder.

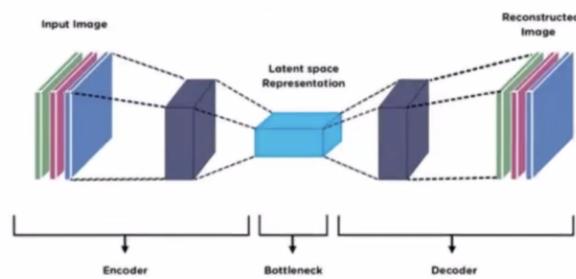


Fig. 2. Estructura de un autoencoder

B. Hiperparámetros a considerar

¿Cuantas capas se deben aplicar?: Se pueden hacer múltiples variables. El tamaño de la codificación (vector latente) va a depender de la arquitectura, entre más capaz mayor comprensión va a tener el modelo, pero al costo de procesamiento.

La profundidad la decide el desarrollador en el diseño, se debe tomar en cuenta en cuenta que a mayor profundidad mayor complejidad.

Función de reconstrucción: es dependiente de la entrada y salida esperada, se usa MSE para determinar qué tan buena es la reconstrucción.

C. Tipos de autoencoder

- Undercomplete autoencoder: reduce la dimensionalidad y aumenta la capacidad de generalización al obligar al modelo a enfocarse en los patrones más importantes.

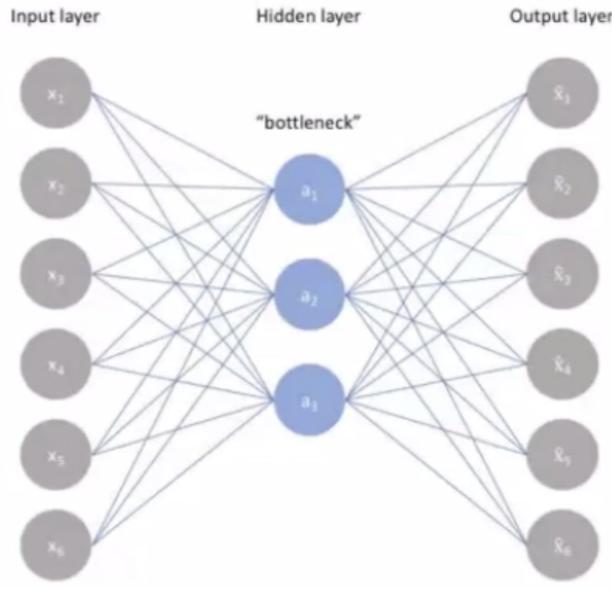


Fig. 3. Diagrama de undercomplete autoencoder

- Sparse autoencoder: mantiene las mismas dimensiones, se pasan apagando neuronas para que aprendan a reconstruir la información de la imagen durante toda la arquitectura.

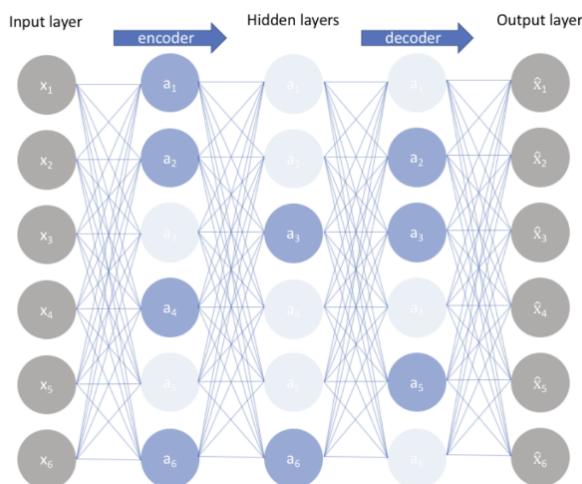


Fig. 4. Diagrama de sparse autoencoder

- Denoising autoencoder: hace que el autoencoder elimine el ruido y en general que haga una tarea en específica en

la cual debe reconstruir una versión limpia a partir de la entrada. Al final el autoencoder aprende a hacer la tarea y el refuerzo es la entrada (X).

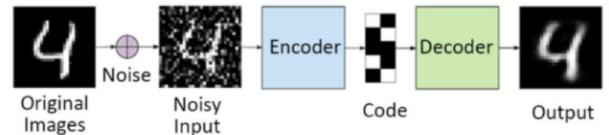


Fig. 5. Diagrama de denoising autoencoder

- Variational autoencoder: se quisiera que ese vector latente (el que se va a construir) siguiera una distribución normal para poder interpolar ciertos valores que hay en esa distribución y obtener nuevos datos.

Se menciona un ejemplo, de cuando se tiene un vector latente lo que se tiene es cada una de los features semánticos, cada uno representar algo de la imagen, si esos features fueran distribuciones para seguir una probabilidad (como se muestra en la figura 6), de esta forma si se agarra uno de esos valores se pueda extrapolar.

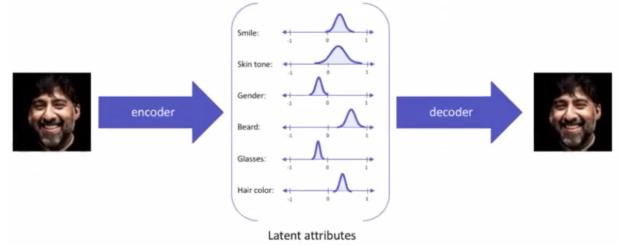


Fig. 6. Ejemplo features de imagen como distribuciones

Eso hace que se puedan obtener los resultados que se ven en la figura 7, existe un deslizamiento suave, conforme se desliza se empiezan a tener imágenes con pequeñas variaciones pero que todavía tienen sentido.

En el variational autoencoder lo primero que se hace es tratar de obtener una distribución o sea un μ y la σ estándar a partir de la imagen. Luego se toman esas muestras para producir el vector latente y finalmente se toma el vector latente para alimentar el decoder.

Lo otro es que se debe calcular la KL divergencia, la cual mide qué tan diferente es una distribución de probabilidad respecto a otra, en nuestro caso se compara una distribución normal estándar (tiene una media 0 y desviación estándar 1). En clase se mencionó un ejemplo: las alturas de las personas, y se quiere determinar si la distribución normal estándar se ajusta más a la distribución A o B, entre menor distancia haya entre los datos menor la divergencia, esto permite ver si falta o se pierde información entre distribuciones.

- U-Net: su tarea es tratar de hacer segmentación de imágenes. En general el concepto de autoencoder permanece sólo que introduce skip connections entre el

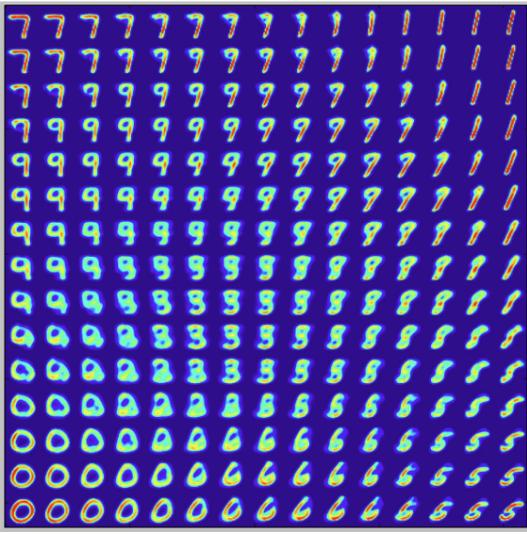


Fig. 7. Interpolación en el espacio latente de un autoencoder

encoder y decoder, estos le permite conservar detalles importantes durante la reconstrucción. Se menciona el paper U-Net: Convolutional Networks for Biomedical Image Segmentation.

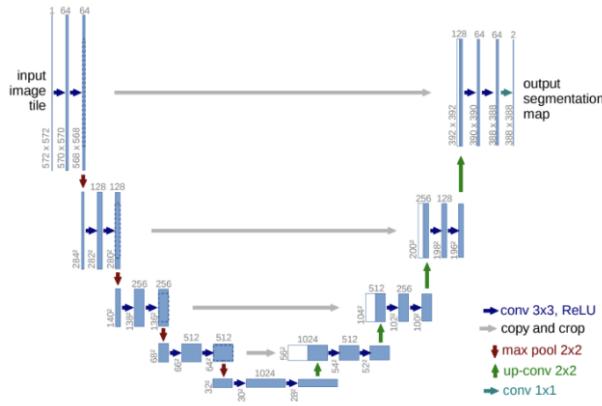


Fig. 8. Diagrama de u-net

III. QUANTIZATION

Modelos más grandes como los de lenguajes, tienen una cantidad de parámetros muy grande, consume muchos recursos, luego viene la intriga de cómo se carga a memoria, en general todos los pesos son cálculos de punto flotante son bastantes lentas en hacer operaciones de punto flotante.

La solución a esto es la quantization, la idea es reducir el número de bits que se tiene para representar los parámetros, para eso las representaciones que están en punto flotante se pasan a entero, por lo que se pierde precisión pero en este caso es el precio a pagar pensando en optimizar el tiempo. Hay formas de compensación de un modelo de 8, 5, 2 o 1 bit.

Nota aclaratoria: quantization no significa aplicar sólo técnicas de redondeo a nuestros pesos y convertirlos a enteros,

la técnica tiene otros cálculos más complejos además de esa transformación.

A. Ventajas

- Disminuye el consumo de memoria al momento de cargar los modelos, además de que se puede usar en devices
- Mejor tiempo de inferencia ya que son datos más simples
- Se tiene un menor consumo de energía, porque se hace menos procesamiento

IV. ¿CÓMO SE REPRESENTAN LOS NÚMEROS?

Las computadoras usan un número fijo para representar los datos. Se puede representar 2^N números distintos en una secuencia de bits como se muestra en la figura 9 en donde $N=3$. Generalmente los bloques de enteros son de 8 bits (byte).

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

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

Fig. 9. Representación de 2^N números distintos en una secuencia de bits

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 en caso de ser positivo
 - El complemento en caso de ser negativo
- La ventaja de esto es que se puede tener una única representación del cero.

A. ¿Cómo se representan los puntos flotantes?

Básicamente incluyen números elevados al negativo de la base. Ejemplo:

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

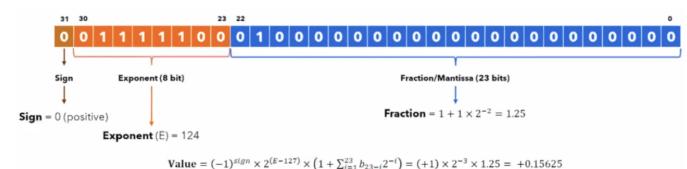


Fig. 10. Representación de un número en formato punto flotante IEEE 754 de 32 bits

En la figura 10 se muestra la representación de un número punto flotante usando 32 bits, como se mencionó anteriormente el primer bit representa el signo, los siguientes 8 bits son para el exponente que se elige, y los 23 bits restantes son para la fracción (Mantissa). Al final se obtiene que el número decimal es +0.15625.

B. Redes neuronales - pesos

Todas las matrices de pesos son representadas con punto flotante, incluyendo los bias, la idea es tratar de no perder calidad en los resultados cuando se haga quantization.

- Lo que se quiere es procesar todas estas operaciones utilizando aritmética de enteros.
- Cuantizar: X, W, B
- Decuantizar: para procesar la siguiente capa

Uno de los principios que se tiene es que las siguientes capas tienen que recibir los valores en punto flotante, entonces los integers de salida se pasan de nuevo a flotante, a este paso se le conoce como dequantization. Las capas siguientes no tienen que darse cuenta que las operaciones hechas se hicieron con enteros.

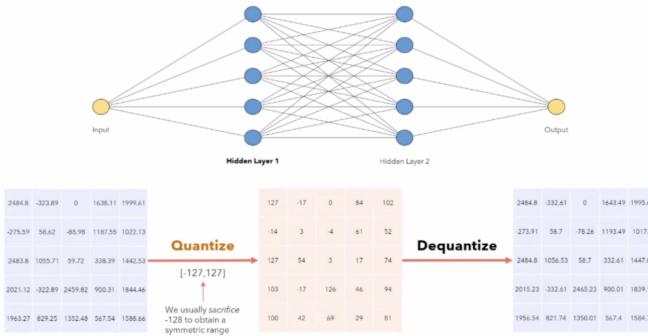


Fig. 11. Ejemplo de red

En la figura 11 se puede observar que después de realizar quantization se pierden ciertos datos, y hay otros que se mantienen, es parte del proceso.

V. TIPOS DE QUANTIZATION

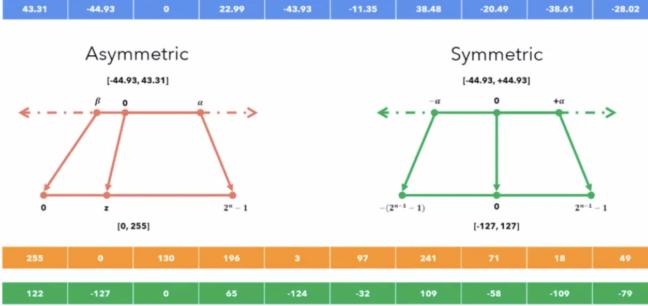


Fig. 12. Quantization simétrica y asimétrica

A. Asimétrico

En el asimétrico no hay valores negativos, sólo hay valores entre 0 hasta el mayor número posible con respecto a los bits que se están aplicando. Se parametriza a un valor específico z, este es el nuevo 0.

Acá se va a intentar mapear los valore en un rango específico $[0, 2^N - 1]$ para esto se deben sacar dos valores:

- $\beta =$ el número menor de los valores a convertir

- $\alpha =$ el número mayor de los valores a convertir

Si se están usando 8 bits se puede representar un total de $[0, 255]$.

¿Cómo se convierten los datos a un vector quantizado?

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

Donde:

- $x_f =$ valor flotante, es el valor del tensor sin quantizar
- $s = \frac{\alpha - \beta}{2^n - 1}$, es el factor de parametrización, es en cuánto se va a escalar los datos
- $z = [-1 \cdot \frac{\beta}{s}]$, es el desplazamiento con respecto al cero
- n = número de bits

El dequantization asimétrico se hace con la siguiente fórmula:

$$x_f = s(x_q - z)$$

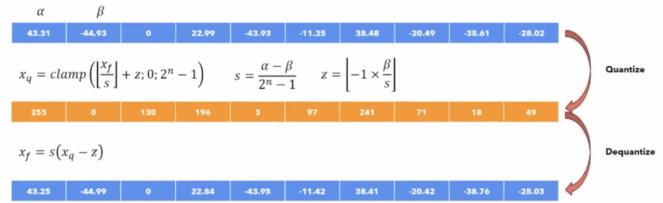


Fig. 13. Quantize y dequantize asimétrico

En la figura 13 se ven las fórmulas aplicadas tanto de quantize y dequantize, se puede observar que los valores pierden precisión.

B. Simétrico

En el caso del quantization simétrico, si hay valores positivos y negativos, contrario al asimétrico su cero si se mantiene.

Se mapean los valores entre un rango $[-\alpha, \alpha]$ a un rango $[-(2^n - 1), (2^n - 1)]$, en este caso:

- $a =$ mayor número absoluto

Si se están usando 8 bits se puede representar un total de $[-127, 127]$.

Fórmula para aplicar quantization simétrico:

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

Donde:

- $x_f =$ valor flotante, es el valor del tensor sin quantizar
- $s = \frac{\text{abs}(\alpha)}{2^n - 1}$, es el parámetro de escalado
- n = número de bits

El dequantization simétrico se hace con la siguiente fórmula:

$$x_f = sx_q$$

En la figura 14 se ven las fórmulas aplicadas tanto de quantize y dequantize, notar que el α se convierte en -127

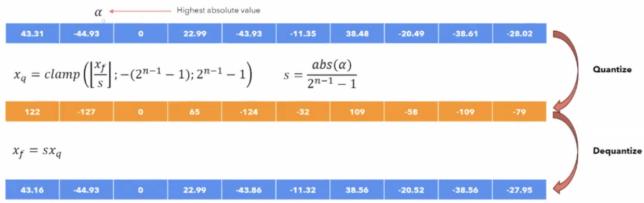


Fig. 14. Quantize y dequantize simétrico

que es el valor más pequeño que se puede representar usando 8 bits. Además de que el cero se mantiene igual.

Al final tanto el quantization asimétrico como el simétrico introduce al error.

VI. DYNAMIC QUANTIZATION

Teniendo los valores de W y B, se debe convertir el X, lo que pasa es que esos X empiezan a variar ese quantization se puede calcular de manera dinámica cada vez que el valor pasa por el Y, a eso se le conoce como dynamic quantization.

Se debe hacer dequantization de cada etapa, pero ¿cómo se hace si no se tiene el parámetro escalador ni su cero?, para eso se puede aplicar Calibration.

A. Calibration

Calibration lo que trata de hacer es capturar estadísticas típicas en cada una de las capas, para a partir de esos datos obtener un α y β razonables para calcular el escalador s y el valor z . Transforma el y_q que son enteros a flotantes.

B. Estrategia de selección del rango

- En el caso asimétrico se seleccionan los valores de los extremos del tensor: $[\beta, \alpha]$. Esta estrategia es muy sensible a los outliers, al hacer dequantization funciona muy mal. Hay varias soluciones para esto:
 - 1) Utilizar una estrategia basada en el percentil de la distribución del vector V, con $\alpha = \text{percentil}(V, 99\%)$ y $\beta = \text{percentil}(V, 1\%)$. De esta forma se acota el outlier y se reduce el error.
 - 2) Tratar de hacer una búsqueda como GridSearch entre parámetros de β y α , que lo que hagan sea minimizar el MSE.
 - 3) Para LLM hay otras estrategias que se aplican como: Softmax Layer LLM, Greedy y Top-P.
- En el caso simétrico se selecciona el mayor valor en términos absolutos: $[-\alpha, \alpha]$

C. Quantization Granularity (Convolución)

Esto funciona un poco diferente en la parte de convolución, recordar que las convoluciones están hechas por muchos filtros, lo que pasa es que esas características son propias del filtro, por lo que siguen una distribución específica, tiene valores mínimos y máximos diferentes, por lo que no se puede aplicar el mismo valor de β y α para todos los filtros. Si se aplica por layer se tienen muchos problemas, si se aplica por canal se resuelve el problema.