R4-模型的高效服務和量化

黃建中, 黃郁雯, 李宜昀, 蔡依儒

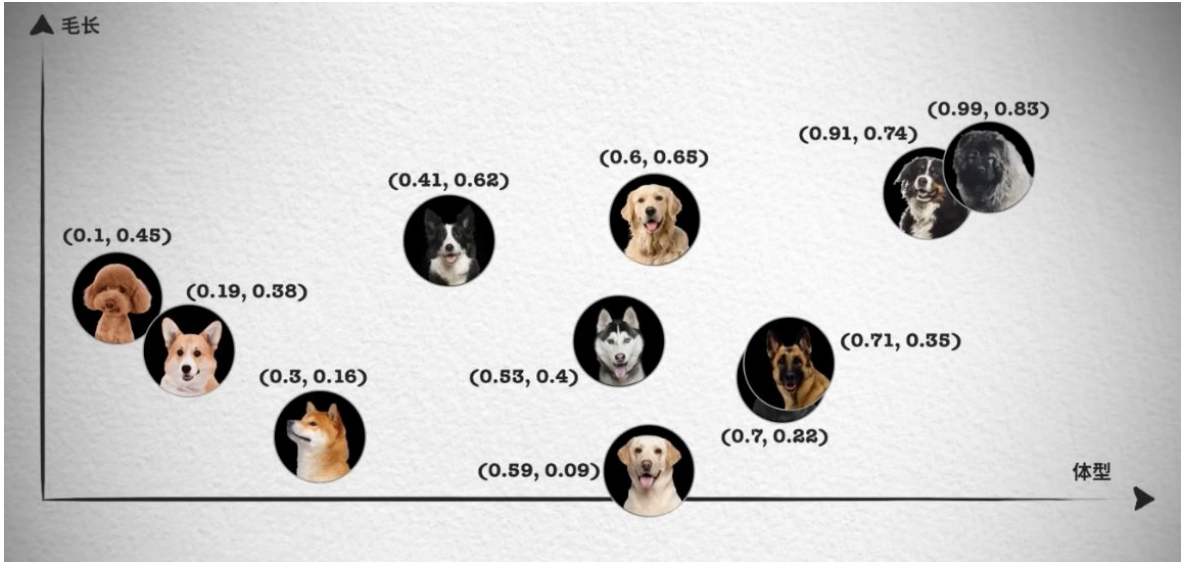
Agenda

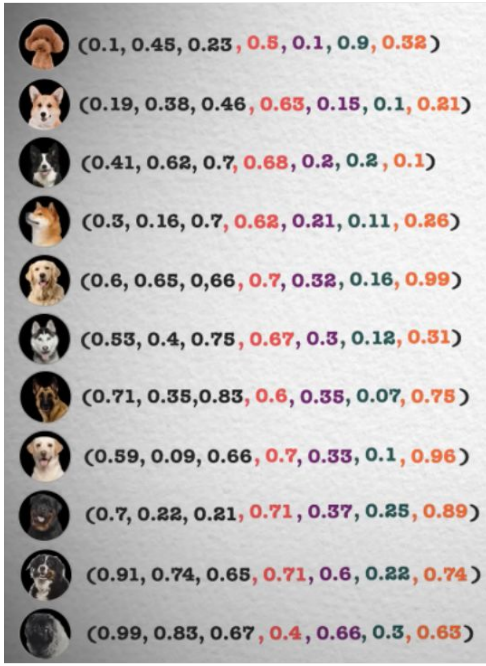
- Vector database 介紹
- 模型壓縮 - quntization
- 模型壓縮 - pruning、distillation、LoRA
- 程式實作

Vector database 介紹

什麼是向量資料庫？

- 向量是由數字組成的數組，通常用來表示文本、圖像、音頻等數據的特徵
- 傳統的關聯式資料庫並不適合儲存和檢索非結構化資料
- 向量資料庫就是一個專門用來儲存和處理向量數據的資料庫

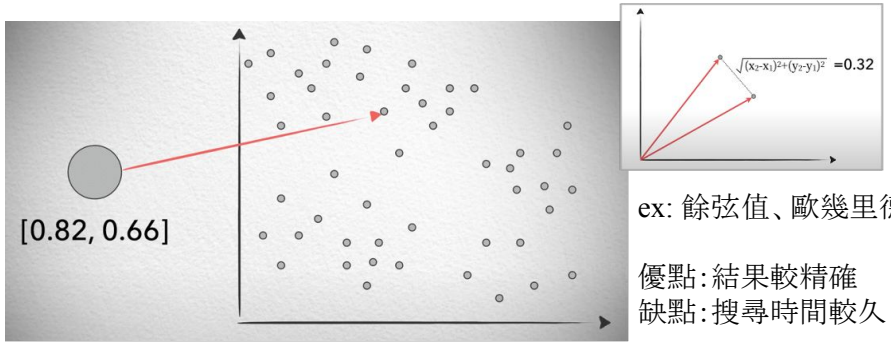




向量資料庫如何運作

和使用查詢語句進行精準搜尋的傳統資料庫不同，向量資料庫使用特定的相似性衡量來尋找最接近的資料。

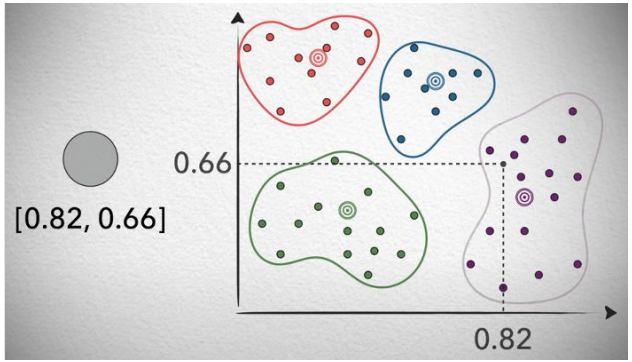
最近鄰搜索 (Nearest Neighbor Search)



ex: 餘弦值、歐幾里德距離

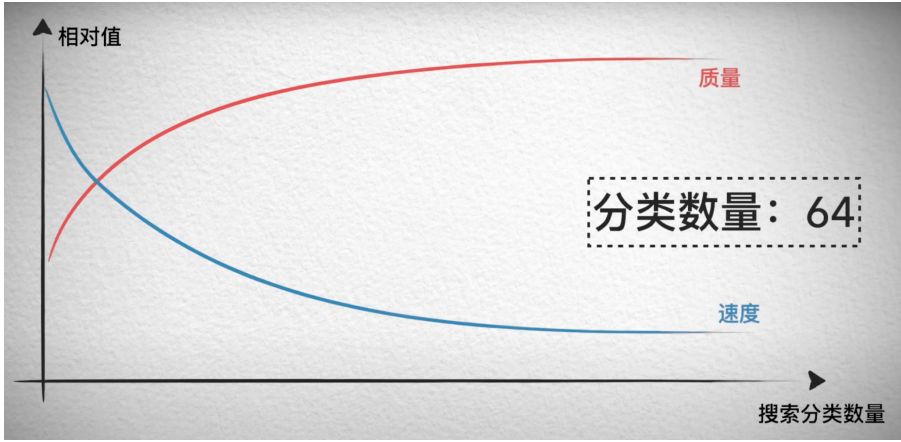
優點:結果較精確
缺點:搜尋時間較久

近似最近鄰搜索 (Approximate Nearest Neighbor)



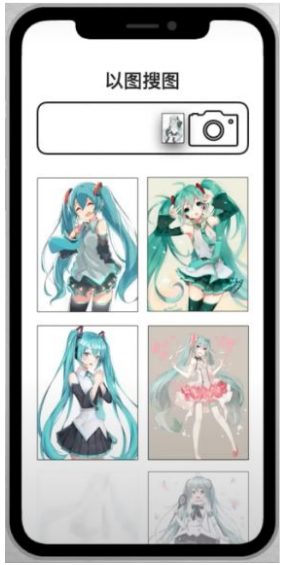
ex: K-means、LSH

優點:搜尋時間較快
缺點:結果較不精確



向量資料庫的應用場景

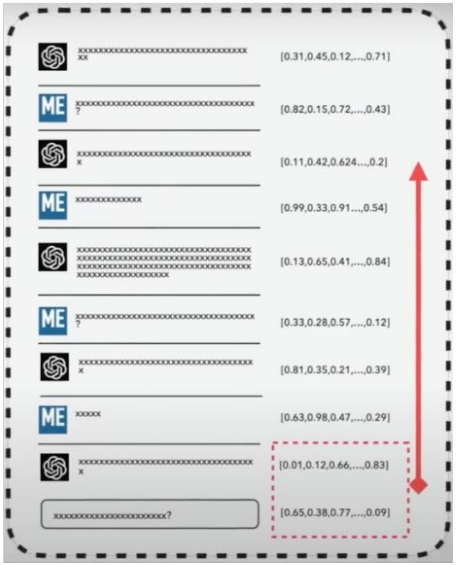
向量資料庫有多種實際應用場景，例如：為大型語言模型提供長期記憶（例如 Lang chain）、基於意義或上下文進行語義搜索、圖像、音訊、影片資料的相似性搜索、排名和推薦引擎（例如為客 戶建議與過去購買相似的商品）等。



相似性搜索

商品推薦

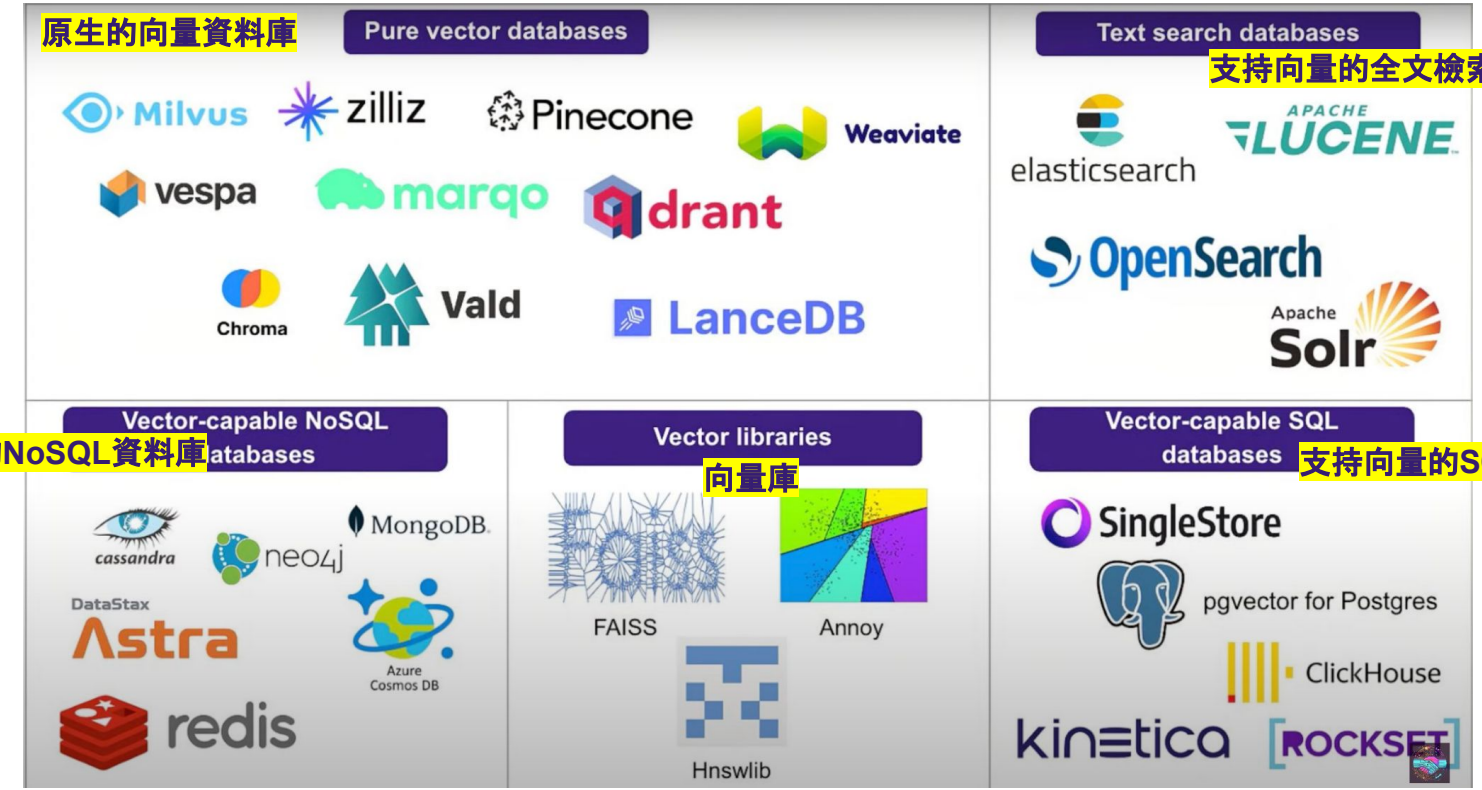
相似問題



長期記憶

向量資料庫的選擇

目前市場上有多種可用的向量資料庫，每個資料庫都有獨特的優勢和功能，可依專案需求選擇適合的資料。



Vector Databases vs. Vector Library

#	Options	Calculation Mechanism	Use cases	Common Vendors/Libraries used in the industry
1.	Vector database	Store the embeddings in a database than query the database	For thousands, millions or billions of vectors that require search on a regular basis.	Qdrant, Pinecone, Zilliz, Weaviate, Chroma, Milvus, Faiss, Langchain memory, Supabase pgvector, Azure Cosmos DB Vector Database Extension, Azure PostgreSQL Server pgvector Extension, Azure AI Search
2.	Vector Libraries	Handling search through in-memory calculations	Primarily used for smaller projects as it provides most of the features needed. Most vendor products can handle vector sets of any size	Facebook Faiss, Spotify Annoy, Google ScaNN, NMSLIB, HNSWLIB

各資料庫特點

向量資料庫	開發商	特點	缺點	開源
Milvus	Zilliz	高性能、高可用、高擴展性	部署和運維可能需要一定的學習曲線	是
Pinecone	NVIDIA	高性能、低延時	雲端服務需要訂閱，可能有成本考量	否
Annoy	Spotify	高性能、易用性	不支援更新或刪除向量，不適合需要高精度的應用	是
Faiss	Facebook	高性能、可擴展性	配置和使用可能相對複雜	是

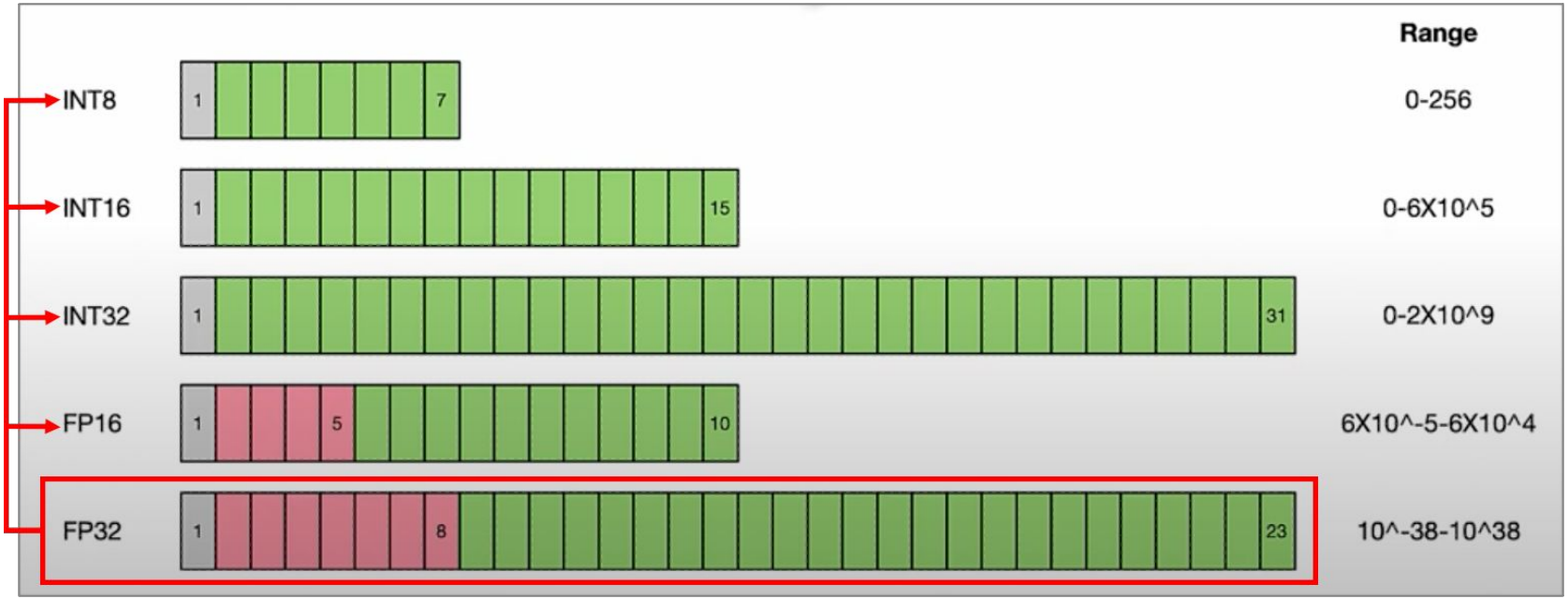
模型壓縮 - quantization

什麼是量化???

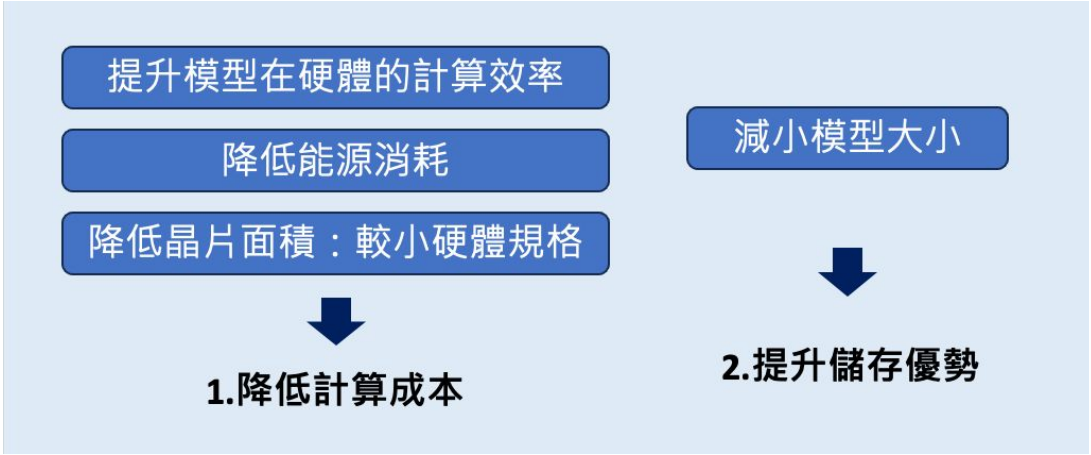
http://tech.sina.com.cn/it/2015-07-04/doc-ifxesftm9556659.shtml

Quantization 量化

是模型壓縮的一種常用方法，通常情況下可以使用不同的量化策略，將模型的參數與運算的精度從浮點數(FP32)降低至較低的精度，如INT8。



為什麼要進行量化？



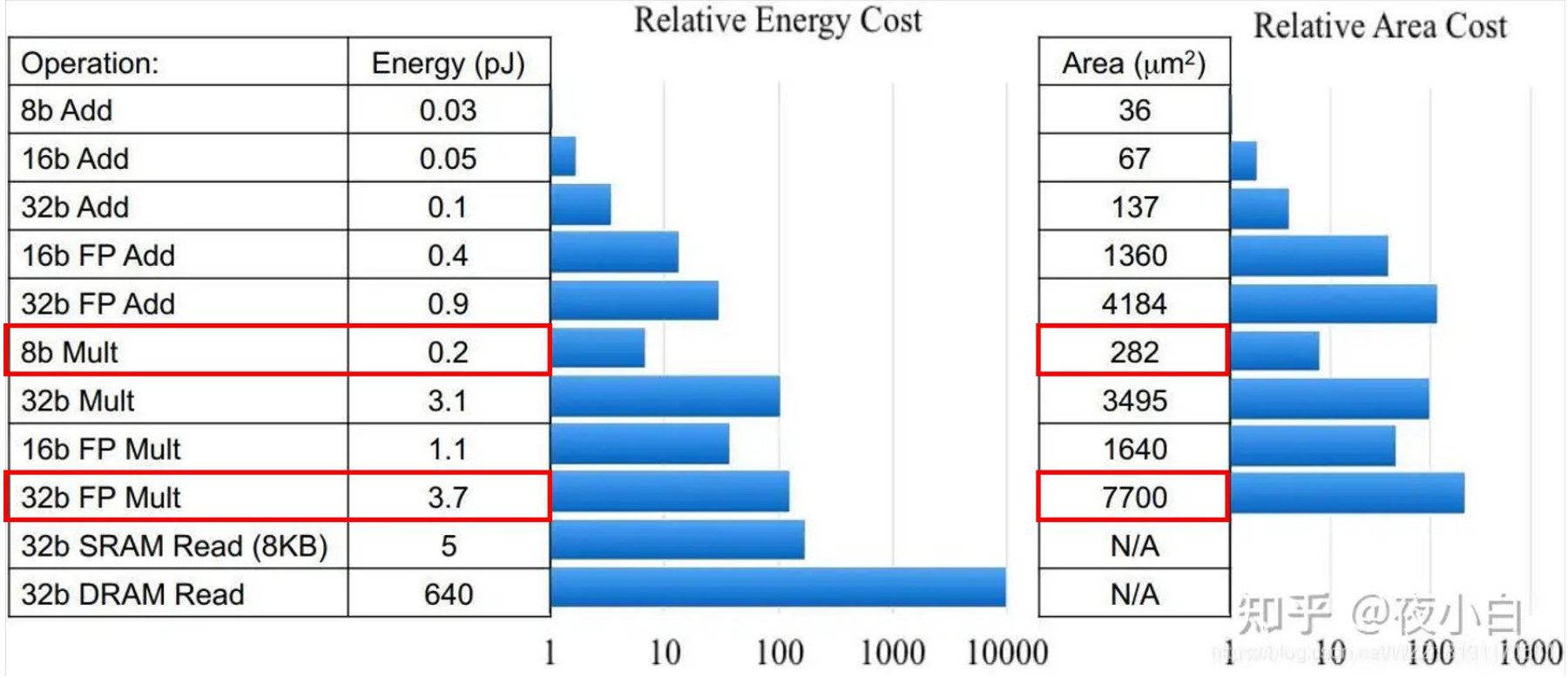
SPECIFICATIONS	
GPU Architecture	NVIDIA Turing
NVIDIA Turing Tensor Cores	320
NVIDIA CUDA® Cores	2,560
Single-Precision	8.1 TFLOPS
Mixed-Precision (FP16/FP32)	65 TFLOPS
INT8	130 TOPS
INT4	260 TOPS
GPU Memory	16 GB GDDR6 300 GB/sec
ECC	Yes
Interconnect Bandwidth	32 GB/sec
System Interface	x16 PCIe Gen3
Form Factor	Low-Profile PCIe
Thermal Solution	Passive
Compute APIs	CUDA, NVIDIA TensorRT™, ONNX

16倍

現在無論是伺服器端還是移動端，INT8 量化都是一個發展趨勢，比如阿里含光、華為 Atlas、華為 NPU、高通 NPU、NVIDIA 的 Tensor Core 等計算設備都支持 INT8 計算。
(右圖：Nvidia T4 GPU上INT8算力是单精度算力的16倍)

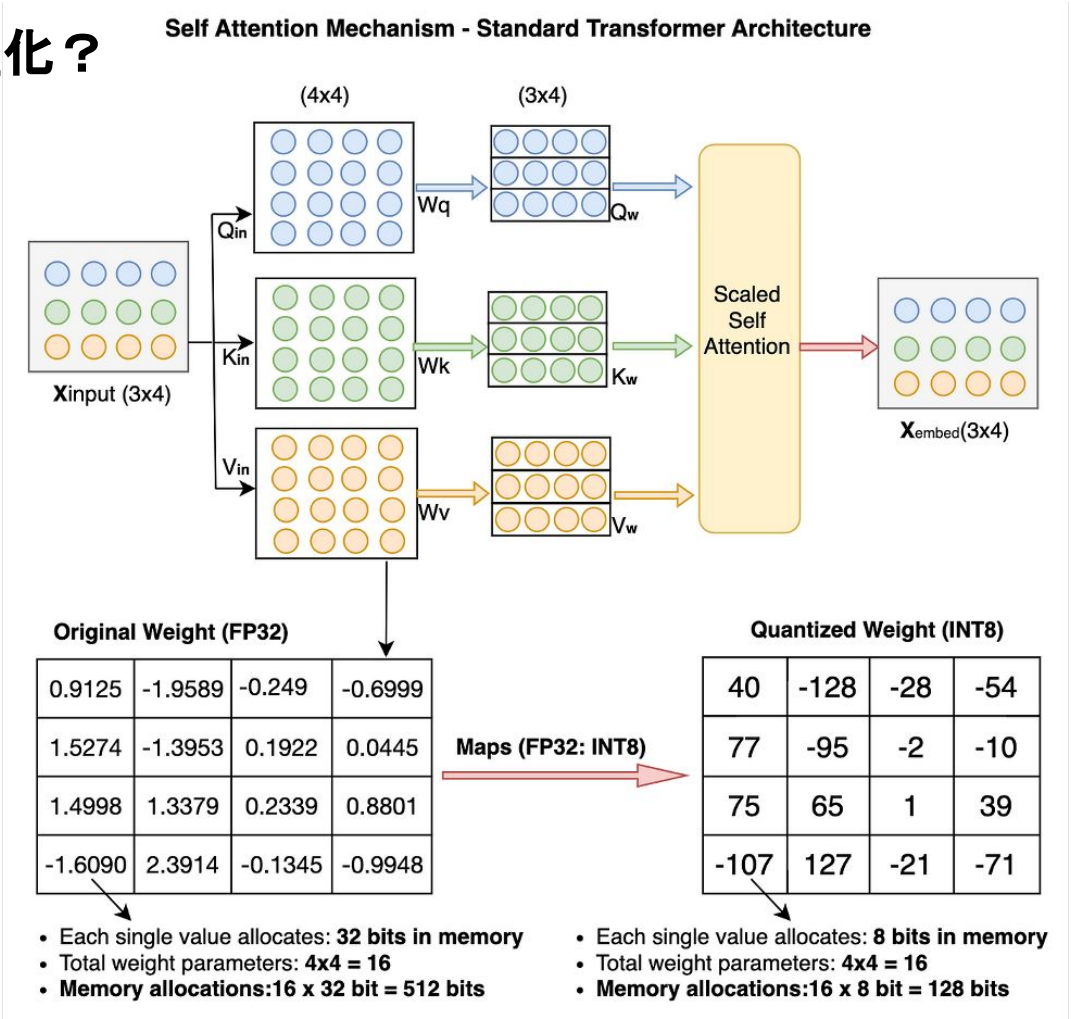
為什麼要進行量化？

Operation:	Energy (pJ)	Area (μm²)
8b Add	0.03	36
16b Add	0.05	67
32b Add	0.1	137
16b FP Add	0.4	1360
32b FP Add	0.9	4184
8b Mult	0.2	282
32b Mult	3.1	3495
16b FP Mult	1.1	1640
32b FP Mult	3.7	7700
32b SRAM Read (8KB)	5	N/A
32b DRAM Read	640	N/A



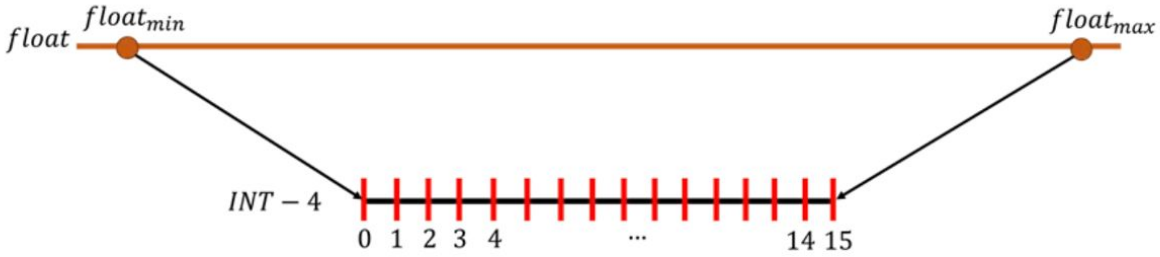
FP32 乘法運算的能耗是 INT8 乘法運算的18.5倍，晶片佔用面積是 INT8 的27.3倍

為什麼要進行量化？


Self Attention Mechanism - Standard Transformer Architecture
(4x4)
(3x4)
Wq
Qw
Qin
Kin
Wk
Kw
Vin
Wv
Vw
Scaled Self Attention
Xinput (3x4)
Xembed(3x4)
Original Weight (FP32)
0.9125 -1.9589 -0.249 -0.6999
1.5274 -1.3953 0.1922 0.0445
1.4998 1.3379 0.2339 0.8801
-1.6090 2.3914 -0.1345 -0.9948
Maps (FP32: INT8)
Quantized Weight (INT8)
40 -128 -28 -54
77 -95 -2 -10
75 65 1 39
-107 127 -21 -71
• Each single value allocates: 32 bits in memory
• Total weight parameters: 4x4 = 16
• Memory allocations:16 x 32 bit = 512 bits
• Each single value allocates: 8 bits in memory
• Total weight parameters: 4x4 = 16
• Memory allocations:16 x 8 bit = 128 bits

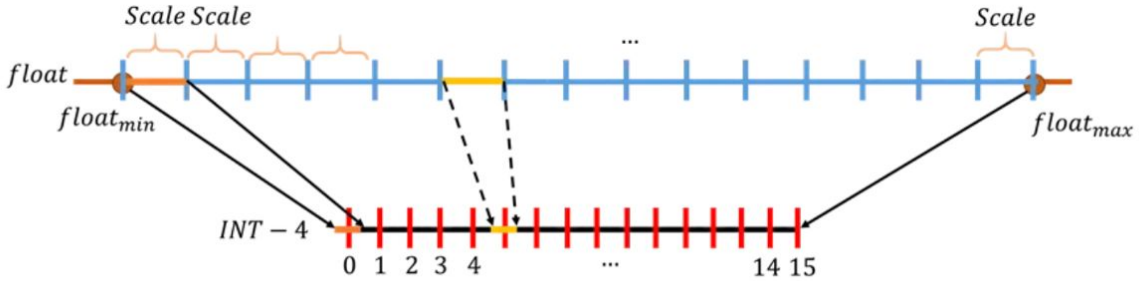
Quantization量化方法簡介（以int4為例）

如果先不考慮負號，最簡單的方式就是把「浮點數的值域最大值轉換成15」，「浮點數的值域最小值轉換成0」，也就是下圖。



INT4的Quantization轉換。

介紹最簡單的方式，就是每一格大小都一樣，所以只要把浮點的值域(最大值-最小值)除上15格，就可以了，而在浮點數上每一格子的大小也稱為Scale，見下圖。



浮點數上的間隔是一樣寬的。

Quantization量化方法：對稱(Symmetric)量化和非對稱(Asymmetric)量化

上面範例介紹的是浮點數轉換到0~15，但整數其實是有正負值的，也就是INT4其實也可以是(-8, -7,-6,-5,-4,-3,-2,-1, +0, +1, +2, +3, +4, +5, +6, +7)，也就是第一個bit是正負號。

假設浮點數的模型權重是:

Float(x_f)	0.1	0.2	1.2	3	2.1	−2.1	−3.5

對稱(Symmetric)量化　-以int4為例：值域為-8~7，共16個數字

1. 觀察浮點數資料的取絕對值後的最大值:

$$range_{x_f} = \max(|x_f|)$$

$$range_{x_f} = \max(|[0.1, 0.2, 1.2, 3, 2.1, -2.1, -3.5]|) = 3.5$$

2. 計算Scale

$$\alpha_x = \frac{2^{(n_{bit}-1)} - 1}{range_{x_f}}$$

$$\alpha_x = \frac{2^{(n_{bit}-1)} - 1}{range_{x_f}} = \frac{2^{(4-1)} - 1}{3.5} = \frac{7}{3.5} = 2$$

3. 量化: 浮點數到定點

$$x_q = round(\alpha_x x_f)$$

$$x_q = round(2 \times [0.1, 0.2, 1.2, 3, 2.1, -2.1, -3.5])$$

$$= round([0.2, 0.4, 2.4, 6, 4.2, -4.2, -7]) = [0, 0, 2, 6, 4, -4, -7]$$

非對稱(Asymmetric)量化　-以int4為例：值域為0~15，共16個數字

1. 觀察浮點數資料的值域:

$$range_{x_f} = \max(x_f) - \min(x_f)$$

$$range_{x_f} = \max([0.1, 0.2, 1.2, 3, 2.1, -2.1, -3.5]) - \min([0.1, 0.2, 1.2, 3, 2.1, -2.1, -3.5]) = 3 - (-3.5) = 6.5$$

2. 計算Scale

$$\alpha_x = \frac{2^{(n_{bit})} - 1}{range_{x_f}} \qquad \alpha_x = \frac{2^{(n_{bit})} - 1}{range_{x_f}} = \frac{2^{(4)} - 1}{6.5} = \frac{15}{6.5} \approx 2.3077$$

3. 計算zero-point

$$zp_x = round(\min(x_f) \times \alpha_x)$$

$$zp_x = round(\min(x_f) \times \alpha_x) = round(-3.5 \times 2.3077) = round(-8.07695) = -8$$

4. 量化浮點數到定點

$$x_q = round(\alpha_x x_f) - zp_x$$

$$x_q = round(2.3077 \times [0.1, 0.2, 1.2, 3, 2.1, -2.1, -3.5]) - (-8)$$
$$= [8, 8, 11, 15, 13, 3, 0]$$

量化效果怎麼衡量：de-quantization

對稱(Symmetric)量化

$$\hat{x}_f = \frac{x_q}{\alpha_x} = \frac{[0, 0, 2, 6, 4, -4, -7]}{2} = [0.5, 0.5, 1, 3, 2, -2, -3.5]$$

非對稱(Asymmetric)量化

$$\hat{x}_f = \frac{x_q + zp_x}{\alpha_x} = \frac{[8, 8, 11, 15, 13, 3, 0] + (-8)}{2.3077}$$
$$= [0, 0, 1.3, 3.033, 2.167, -2.167, -3.467]$$

整合一下對稱和非對稱的總誤差值，如下

Float(x_f)	0.1	0.2	1.2	3	2.1	−2.1	−3.5	總和
對稱 還原後誤差	0.1	0.2	0.2	0	0.1	0.1	0	0.7
非對稱 還原後誤差	0.1	0.2	0.1	0.033	0.067	0.067	0.033	0.6

量化還原後誤差越小代表量化效果越好

按量化的過程是否要進行訓練可以分成兩類

QAT（訓練時量化）	PTQ（訓練後量化）
需要重新訓練模型	不需要重新訓練模型
由於量化參數是通過微調過程學習得到，通常能夠保證精度損失較小	訓練與量化過程沒有關聯，通常難以保證量化後模型的精度

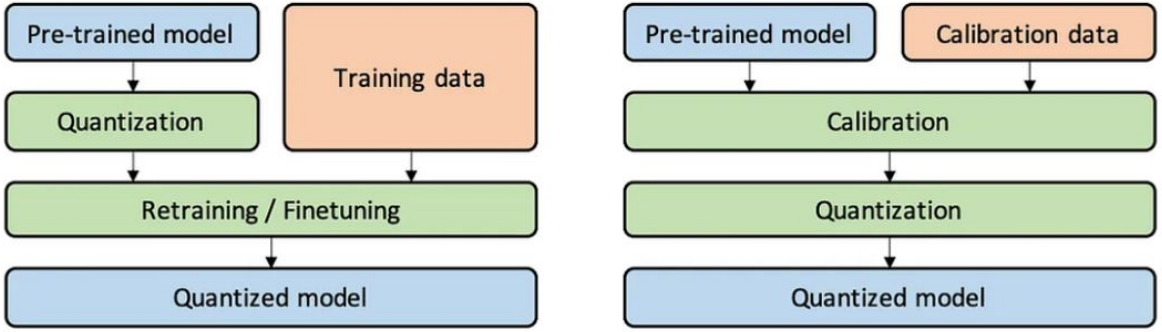
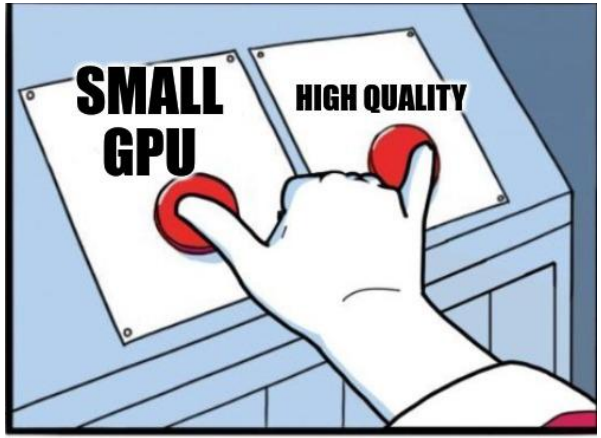


Figure 4: *Comparison between Quantization-Aware Training (QAT, Left) and Post-Training Quantization (PTQ, Right). In QAT, a pre-trained model is quantized and then finetuned using training data to adjust parameters and recover accuracy degradation. In PTQ, a pre-trained model is calibrated using calibration data (e.g., a small subset of training data) to compute the clipping ranges and the scaling factors. Then, the model is quantized based on the calibration result. Note that the calibration process is often conducted in parallel with the finetuning process for QAT.*


SMALL GPU
HIGH QUALITY


imgflip.com


QUANTIZED MODELS
LOSS IN ACCURACY
imgflip.com

壓縮模型的其他方法：
Pruning剪枝、Distillation蒸餾、LoRA & QLoRA

Pruning剪枝是什麼、哪些場景可用

定義:移除模型中作用/影響力較小的部分，只保留精華內容
目標:最小化剪枝後的模型損失函數

- **剪枝後模型性能會顯著下降**
  如果剪枝過度會導致模型性能顯著下降，故需要在模型壓缩和性能之間取得平衡

- **剪枝方法選擇**
  - **權重連接剪枝**
    >> 計算每個權重連接的重要性，通常用權重絕對值作為衡量標準，將重要性較低的權重連接設為0，適用於需要**壓縮模型體積來加速運算的場景**，如:即時語音辨識

  - **神經元剪枝**
    >> 計算每個神經元的重要性，通常用該神經元輸出的L1範數作為衡量標準，將重要性較低的神經元對應的整個行或列設為0，相當於去掉某些神經元，**適合保持簡單結構即可部署的模型**

最小化剪枝後的模型損失函數

Pruning數學原理

大型模型的损失函数：$L_{large} = \frac{1}{N}\sum_{i=1}^{N} l(y_{large}^{(i)}, \hat{y}_{large}^{(i)})$

剪枝后的模型损失函数：$L_{small} = \frac{1}{N}\sum_{i=1}^{N} l(y_{small}^{(i)}, \hat{y}_{small}^{(i)})$

剪枝策略：$S(\theta_{large}) = \theta_{small}$

$$\min_{\theta_{small}} L_{small}$$

交叉熵（Cross Entropy）是Shannon信息论中一个重要概念，主要用于度量两个概率分布间的差异性信息。在信息论中，交叉熵是表示两个概率分布 p, q 的差异，其中 p 表示真实分布，q 表示预测分布，那么 $H(p, q)$ 就称为交叉熵：

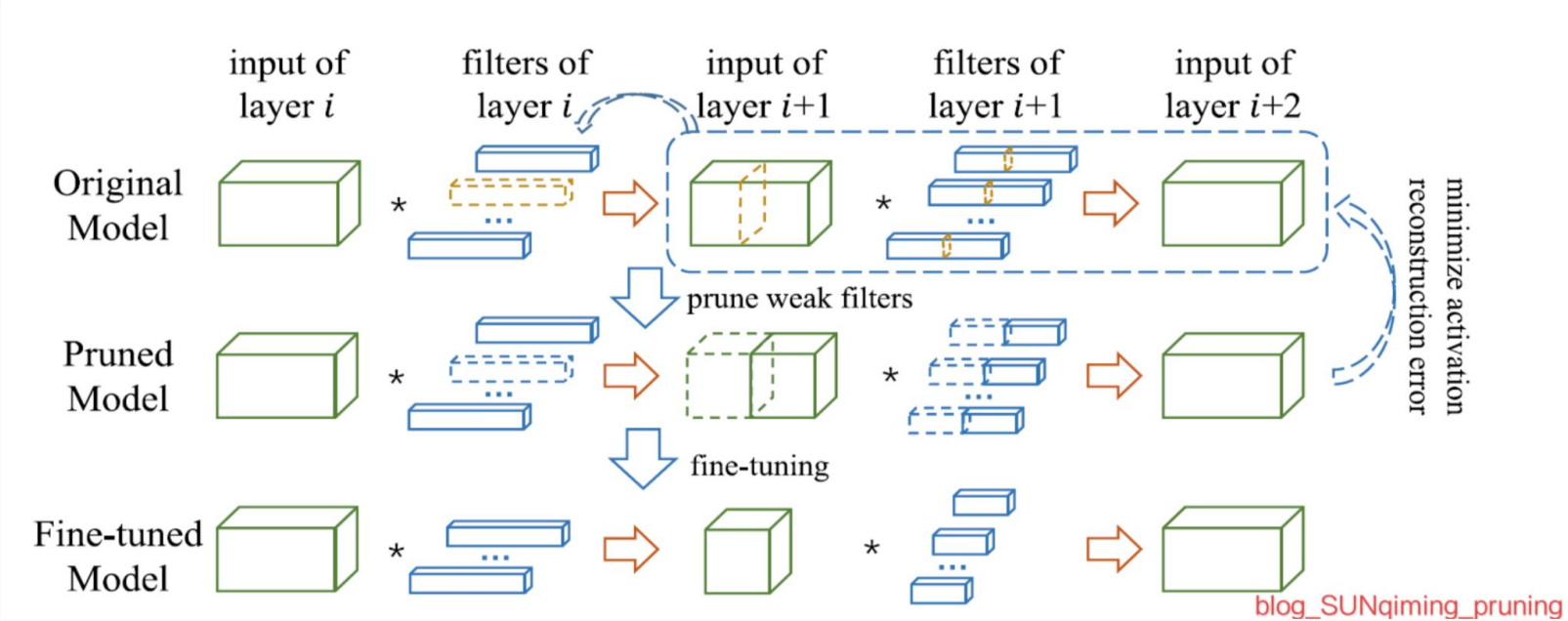
$$H(p,q) = \sum_i p_i \cdot \ln \frac{1}{q_i} = -\sum_i p_i \ln q_i \tag{1}$$

交叉熵可在神经网络中作为损失函数，p 表示真实标记的分布，q 则为训练后的模型的预测标记分布，交叉熵损失函数可以衡量 p 与 q 的相似性。

交叉熵函数常用于逻辑回归(logistic regression)，也就是分类(classification)。

Ref: 數學原理

Pruning剪枝後的模型 VS. 其他模型



Ref: https://blog.csdn.net/weixin_49457347/article/details/117250011, 數學原理

Pruning剪枝副作用

定義:移除模型中作用/影響力較小的部分，只保留精華內容

- **剪枝後的模型加速**
  剪枝後模型會出現大量的稀疏權重 *，故需要使用稀疏矩陣的相關知識来加速運算，否則剪枝的加速效果有限
  *指矩陣中絕大部分元素都是0，只有少數元素不為0的矩陣

- **剪枝對模型可解釋性的影響**
  剪枝會改變模型結構，故有可能影響模型的可解釋性，需要在壓缩和可解釋性之間權衡

Distillation蒸餾法是什麼、可以怎麼用

定義：將大模型的知識蒸餾到小模型中，讓小模型擁有大模型的性能

目標：最小化小型模型的損失函數，同時最小化知識遷移損失

- **優點**
  - **自由度很大**，可以蒸餾出最终的機率結果或是中間層的结果，如：詞向量層、隱層、注意力矩陣
  - 蒸餾後的小模型，**通常要比直接用相同模型訓練後的效果更好**
  - **在保持模型性能與準度的前提下**，**大幅壓缩了模型的體積**，減少訓練成本以外，也讓模型能 夠更容易部署在mobile端和edge端的設備上

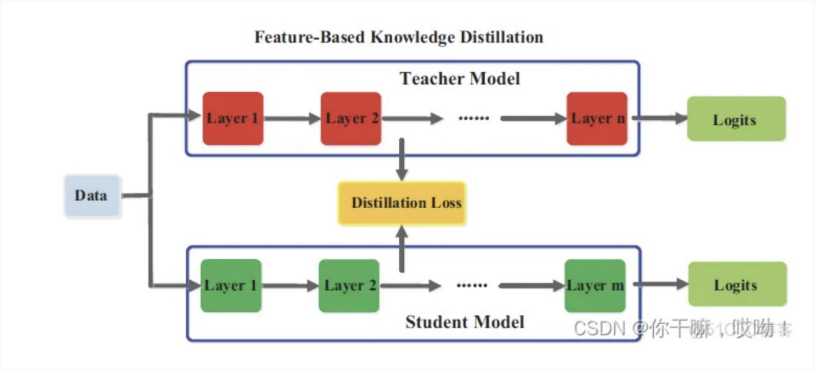
- **應用**
  - **機器學習 - 跨模型知識遷移**：大模型的知識蒸餾到不同架構的小模型中
    例如：將BERT知識蒸餾到BiLSTM或邏輯回歸模型中
  - **多任務學習**：將多個任務的知識蒸餾到一個共享模型中，提升模型的多任務性能
  - **模型壓縮、加强單一模型的性能**

Distillation蒸餾法 - 最佳層對應策略舉例

在大幅減少模型參數的同時，仍能保持接近老師模型的性能，這就是最佳層對應策略的核心思想。

假設老師模型BERT有12個隱藏層，而我們要將其蒸餾為一個 6層的學生模型。最佳層對應策略如下，能讓 6層的學生模型充分吸收 12層老師模型的知識

- 學生模型第1層對應老師模型第1、2層的平均
- 學生模型第2層對應老師模型第3、4層的平均
- 學生模型第3層對應老師模型第5、6層的平均
- 學生模型第4層對應老師模型第7、8層的平均
- 學生模型第5層對應老師模型第9、10層的平均
- 學生模型第6層對應老師模型第11、12層的平均



另個例子：華為的 TinyBERT模型，即使用全面蒸餾法，對 BERT 的詞向量層、中間隱層、注意力矩陣都進行全面的蒸 餾

>>> 實驗结果顯示：性能優於前代的 DistilBERT模型 和 BERT-PKD 模型

最小化小型模型的損失函數，同時最小化知識遷移(KD)損失

Distillation數學原理

大型模型的损失函数：$L_{large} = \frac{1}{N}\sum_{i=1}^{N} l(y_{large}^{(i)}, \hat{y}_{large}^{(i)})$

小型模型的损失函数：$L_{small} = \frac{1}{N}\sum_{i=1}^{N} l(y_{small}^{(i)}, \hat{y}_{small}^{(i)})$

知识迁移损失：$L_{KD} = \frac{1}{N}\sum_{i=1}^{N} l(y_{large}^{(i)}, \hat{y}_{small}^{(i)})$

$$\min_{\theta_{small}} \frac{1}{N}\sum_{i=1}^{N} l(y_{small}^{(i)}, \hat{y}_{small}^{(i)}) + \lambda L_{KD}$$

其中，θ_{small}是小型模型的参数，λ是一个超参数，用于平衡模型性能和模型规模之间的权衡。

交叉熵（Cross Entropy）是Shannon信息论中一个重要概念，主要用于度量两个概率分布间的差异性信息。在信息论中，交叉熵是表示两个概率分布 p, q 的差异，其中 p 表示真实分布，q 表示预测分布，那么 $H(p, q)$ 就称为交叉熵：

$$H(p,q) = \sum_i p_i \cdot \ln\frac{1}{q_i} = -\sum_i p_i \ln q_i \quad (1)$$

交叉熵可在神经网络中作为损失函数，p 表示真实标记的分布，q 则为训练后的模型的预测标记分布，交叉熵损失函数可以衡量 p 与 q 的相似性。

交叉熵函数常用于逻辑回归(logistic regression)，也就是分类(classification)。

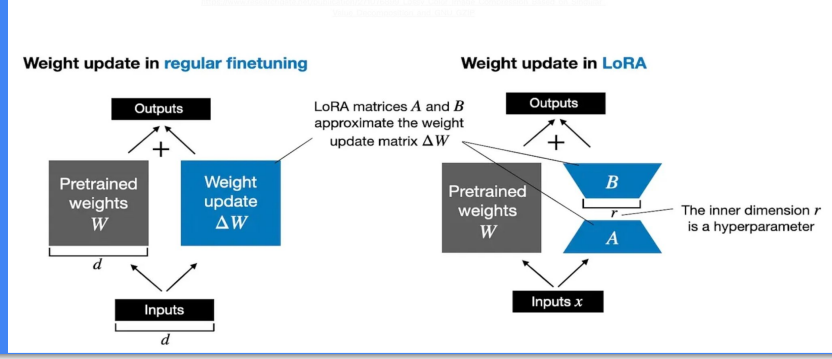
Ref: 數學原理https://juejin.cn/post/7313242014592778267

蒸餾 VS. 剪枝？小朋友才做選擇，我兩個都要

技術趨勢

- **蒸餾+剪枝一起用**：蒸餾可以先將大模型的知識蒸餾到一個小模型中，接著再對這個小模型進行剪枝。如此一來，可以在保持準度的前提下，同時縮小模型體積，並且加速推理運算。
- **動態壓縮模型體積**：根據模型的结構和参數，動態調整壓縮的策略。
- **深度學習模型壓縮**：研究將更複雜的深度學習模型進行壓縮，以適應更多應用場景。
  如：卷積神經網絡、循環神經網路
  - 知識蒸餾的歷史演進：https://blog.csdn.net/abcdefg90876/article/details/109064007
  - 模型蒸馏与模型剪枝：两种模型 压缩方法的融合与 优化(code demo)
    https://juejin.cn/post/7313242014592778267

LoRA是什麼



定義：LoRA（Low-Rank Adaptation）技術是一種用於提升機器學習模型效能的方法，**特別是針對大規模預訓練模型的微調（fine-tuning），主要目標是通過降低微調過程中的參數數量和計算成本，有效地提升模型效能**

基本原理

1. **低秩近似**：是CNN 模型压缩的主流方法之一，基於低秩矩陣分解的概念，將模型的權重矩陣分解成兩個較小的矩陣，降低了微調過程中需調整的參數數量。

2. **保持原始模型權重不變**：在原始模型的基礎上添加適配層（ adapter layers），**這些適配層僅負責低秩近似的更新** 。
   原始模型的權重保持不變，從而 **避免了過度調整導致的過擬合** 。

3. **高效訓練**：由於**僅需要更新適配層的參數**，**LoRA顯著降低計算成本和記憶體使用** ，適合於 **資源有限的環境** 中進行模型微調。
   Ref: 簡單Python Code學習LoRA

LoRA是什麼

應用場景

- **微調大規模模型**：在**大規模預訓練模型**（如BERT、GPT等）進行特定任務的微調時， LoRA可以在**不顯著增加計算資源的前提下**，快速且高效地適應新的任務需求。
- **多任務學習**：
  可以通過**增加不同的適配層**，**每個任務對應一組不同的適配層，這樣可以在共享底層模型的基礎上，同時學習多個任務** 。
- **遷移學習**：
  LoRA技術**特別適合於遷移學習**，在新任務上添加適配層進行微調，快速遷移預訓練模型的知識並應用到新領域。
- **常用場景**：
  - **自然語言處理（NLP):** LoRA被證明可以顯著提升文本分類、命名實體識別（ NER）、機器翻譯等任務的效能
  - **計算機視覺（CV):** 在計算機視覺領域， LoRA技術同樣可以用於圖像分類、目標檢測等任務

QLoRA讓有限的顯卡能力化作無限可能

QLoRA 是一種結合 4bit 量化和 LoRA 微調的高效訓練方法。

它將模型參數量化為 4bit NormalFloat 格式，但在訓練時反量化為 bf16。

利用 LoRA 技術僅只需訓練少量的 LoRA 參數，即可大幅降低顯存需求
例如：33B 參數的 LLaMA 模型可在 24GB 顯卡上訓練。

QLoRA 的關鍵技術：
Block-wise Quantization、Quantile Quantization、雙重量化以及 Page Optimizer 機制。

QLoRA關鍵技術

Block-wise Quantization
將一個很大的數 值範圍分成很多小塊 (block)，每個小塊使用自己的一個常數來壓縮數 值，如此可以避免極端 值影響整體的壓縮。在 QLoRA 中block 大小被設為 64。

Quantile Quantization
假設我們要把數 值量化到 4bit(相當於16 個數字)，最簡單的做法是直接四捨五入
但這樣可能大部分數 值都被量化到同一個數字上，故原本的差異就消失了
Quantile Quantization 的做法是:先把所有數 值排序並分成 16 等份。
第一份映射到 0、第二份映射到 1，以此類推。這樣可以充分利用 16 個數字讓原始數 值在量化後的數字上仍分布均勻

雙重量化
在 Block-wise Quantization 中,每個 block 都有一個常數 c 來壓縮數值。如果 c 用 float32 存,會額外佔用不少顯存。所以 QLoRA 提出了雙重量化:先用 256 個 block 量化一次 c,得到 8bit 的 c'，這樣每個參數只需額外 0.127 bit 就夠了。

Page Optimizer 機制
在顯存不足時可以把優化器 (Optimizer)的狀態暫時放到內存上，需要更新優化器時再從 內存讀回來。
這樣可以大幅降低顯存峰 值的使用。論文稱要在 24GB 顯存上訓練 33B 參數模型，這個機制是必要的。

程式實作