ENTRENAMIENTO DISTRIBUIDO DE MODELOS DE INTELIGENCIA ARTIFICIAL EN ENTORNOS MULTI-GPU

Iñigo Diaz-Munio CDIA + INF
79134735S

OBJETIVOS INICIALES (Primera Entrega)

Objetivo 1.Comprensión teórica y técnica

Estudiar fundamentos del entrenamiento distribuido

Analizar comunicación y sincronización entre GPUs

Entender cuellos de botella de memoria

Objetivo 2: Experimentación práctica (escala reducida)

Implementar y comparar **DOS métodos** de paralelización

Probar diferentes arquitecturas (CNN, MLP, Transformer)

Datasets: MNIST, CIFAR-10, IMDB, WikiText-103

Medir tiempo, uso GPU, precisión

Objetivo 3: Evaluación

Analizar rendimiento y escalabilidad

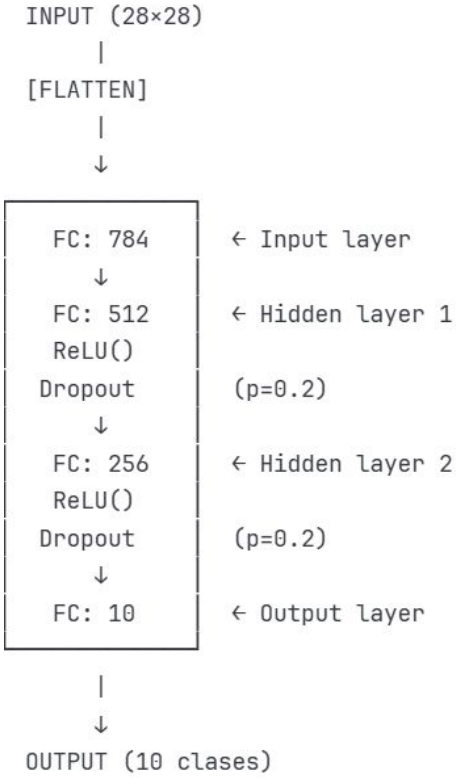
Plantear estrategias de optimización

Reflexionar sobre escalado a multi-GPU real

LO QUE HE HECHO

Coherencia con la propuesta inicial:

Compromiso Inicial	**Estado**	**Resultado**
2 métodos de paralelización	Cumplido	Baseline + DP + DDP
Arquitecturas: CNN, MLP, Transformer	Cumplido	6 modelos diferentes
Datasets propuestos	Cumplido	MNIST, CIFAR-10/100, IMDB, WikiText
Métricas de rendimiento	Cumplido	Todas las propuestas
Una sola GPU (simulación)	Cumplido	



MNIST + MLP

INPUT: Imagen MNIST 28×28 píxeles

FLATTEN (**Aplanar**) Imagen original: [28 × 28] = matriz 2D -> Flatten: [784] = vector 1D .

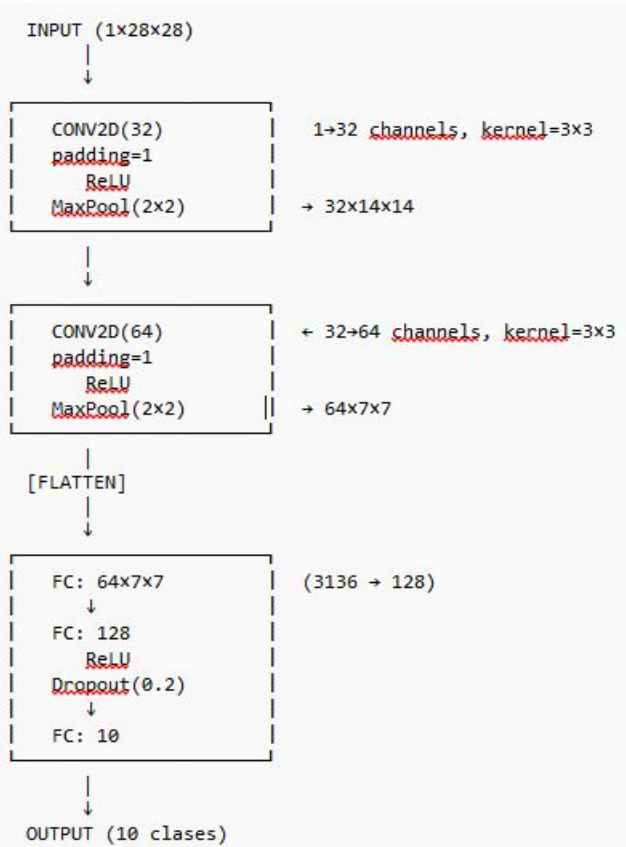
Hidden Layer 1 -> 784 neuronas de entrada. Multiplicación matricial: $W_1 \times x + b_1$ 512 neuronas (aprenden patrones simples). ReLU(x) = max(0, x) y por último,Dropout(0.2) Apaga aleatoriamente 20% de neuronas (para prevenir overfitting).

Hidden Layer 2 -> Mismo proceso.

Output Layer-> Multiplicación: $W_3 \times h_2 + b_3 = 10$

OUTPUT: [0.01, 0.02, 0.05, 0.89, 0.01, 0.01, 0.00, 0.00, 0.01, 0.00]. Más activa -> La 3. (0.89)

¿Qué hace ReLU?
Si activation = -5 → ReLU = 0
Si activation = 0 → ReLU = 0
Si activation = 3 → ReLU = 3 (activa)
Si activation = 10 → ReLU = 10 (activa)



MNIST + CNN Small

INPUT: 1×28×28 (escala de grises)

Bloque 1:

CONV2D(32 filtros, kernel 3×3) → Detecta bordes

ReLU

MaxPool(2×2) → 32×14×14

Bloque 2:

CONV2D(64 filtros, kernel 3×3) → Patrones complejos

ReLU

MaxPool(2×2) → 64×7×7

Clasificador:

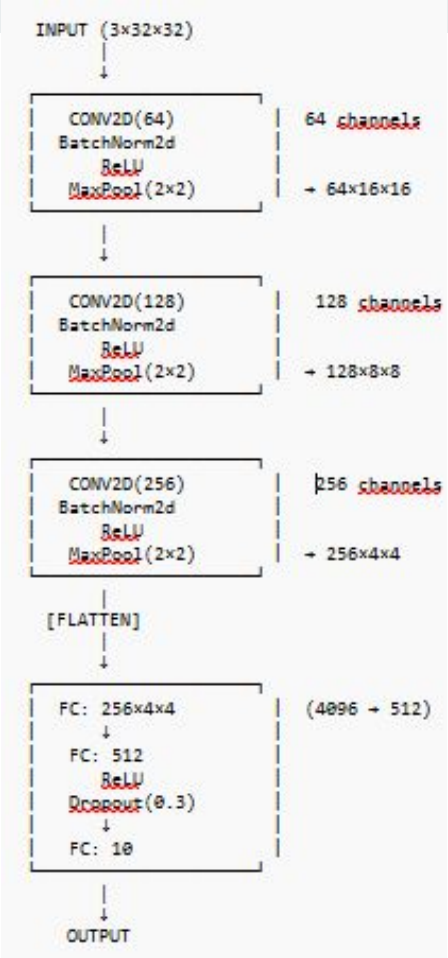
Flatten: 64×7×7 → [3136]

FC: 3136 → 128 + ReLU + Dropout(0.2)

FC: 128 → 10 clases

¿Qué hacen los filtros?

- Filtro 1: Detecta bordes verticales
- Filtro 2: Detecta bordes horizontales
- Filtro 3: Detecta esquinas
- ... (32 filtros en total)



INPUT: 3×32×32 (RGB)

Bloque 1:

CONV2D(64 filtros) → 64×16×16

BatchNorm2d + ReLU + MaxPool

Bloque 2:

CONV2D(128 filtros) → 128×8×8

BatchNorm2d + ReLU + MaxPool

Bloque 3:

CONV2D(256 filtros) → 256×4×4

BatchNorm2d + ReLU + MaxPool

Clasificador:

Flatten: 256×4×4 → [4096]

FC: 4096 → 512 + ReLU + Dropout(0.3)

FC: 512 → 10 clases

Jerarquía de patrones:

Bloque 1: Bordes, colores básicos

Bloque 2: Texturas, formas simples

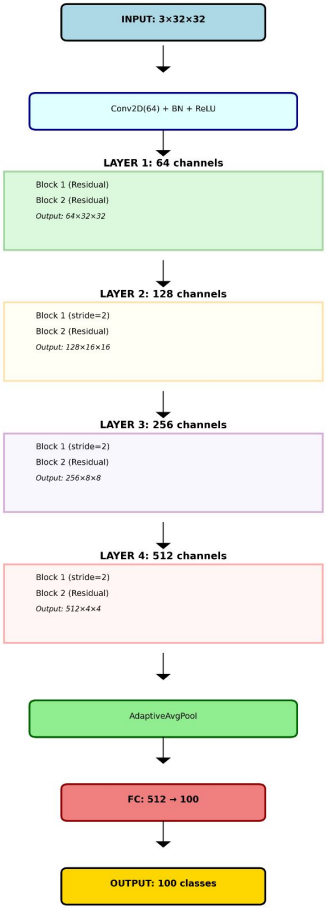
Bloque 3: Partes de objetos (ruedas, alas)

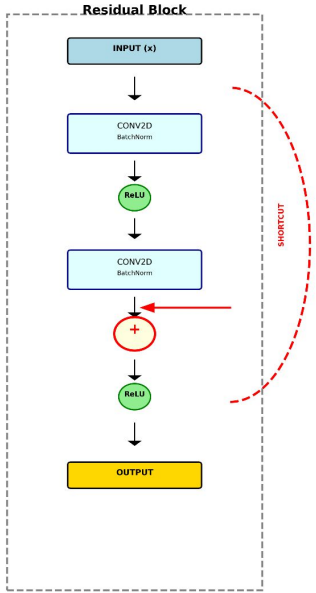
CONV2D aplica un filtro pequeño sobre una imagen para detectar patrones.

BatchNorm normaliza las activaciones para que tengan media=0 y desviación estándar=1. BatchNorm calcula la media y la desviación estandar y luego normaliza cada valor:

valor_norm = (valor - media) / desviación

MaxPool: Reduce el tamaño de la imagen a la mitad tomando el valor máximo de cada matriz 2×2. Esto reduce el tiempo en un 75% y hace el modelo funcione mejor ante pequeños movimientos de la imagen.





ENTRADA: 3×32×32 (imagen RGB de CIFAR-100)

Conv2D(64) + BatchNorm + ReLU (conv inicial para extraer características básicas)

CAPA 1: 64 channels

- Bloque 1 (Residual)
- Bloque 2 (Residual)
- Salida: 64×32×32 (detecta patrones básicos: bordes, colores)

CAPA 2: 128 channels

- Bloque 1 (stride=2) ← reduce tamaño a la mitad
- Bloque 2 (Residual)
- Salida: 128×16×16 (combina patrones básicos en texturas)

CAPA 3: 256 channels

- Bloque 1 (stride=2) ← reduce tamaño a la mitad
- Bloque 2 (Residual)
- Salida: 256×8×8 (detecta partes de objetos)

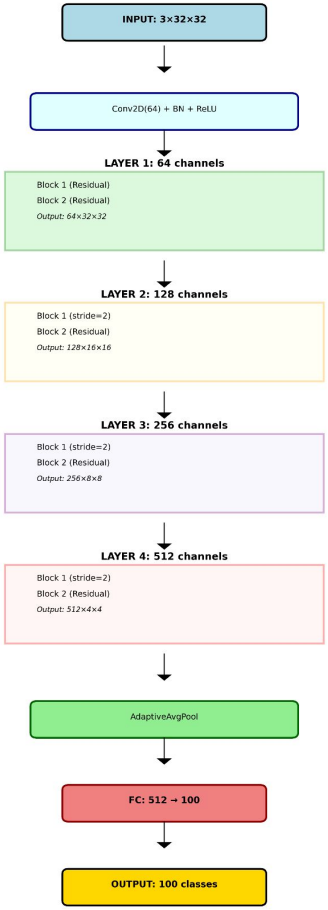
CAPA 4: 512 channels

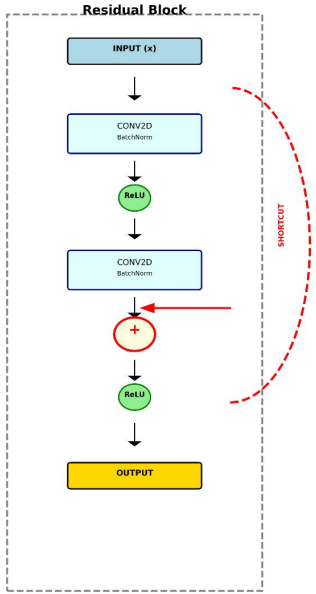
- Bloque 1 (stride=2) ← reduce tamaño a la mitad
- Bloque 2 (Residual)
- Salida: 512×4×4 (representaciones de objetos completos)

AdaptiveAvgPool (reduce a 512×1×1, promediando toda la información espacial)

FC: 512 → 100 (capa totalmente conectada para clasificación)

SALIDA: 100 clases (una probabilidad para cada clase de CIFAR-100)





Residual Block.

El bloque residual permite entrenar redes muy profundas sin que los gradientes se vengan abajo.

Shortcut-> REpresenta el "atajo" que permite que la entrada original (x) se sume directamente a la salida de F(x).

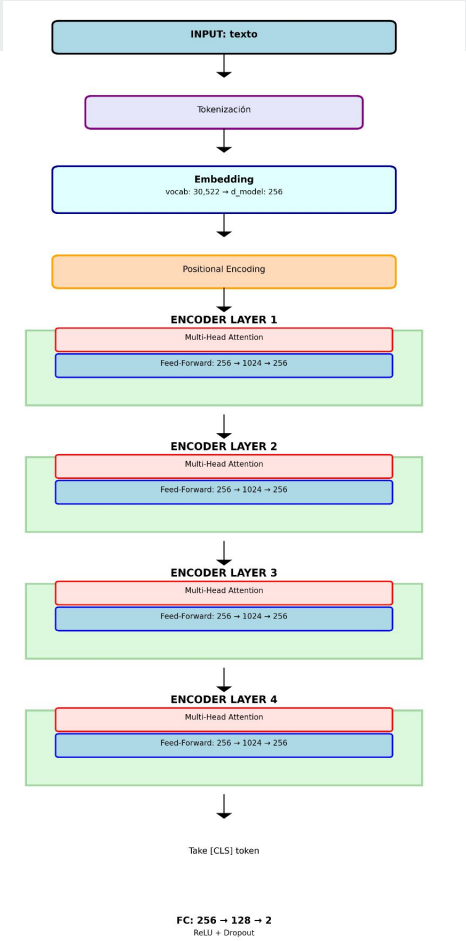
Es como tener un camino principal (el atajo) y un desvío opcional (las convoluciones). Si el desvío no mejora el resultado, sigue por el camino principal.

Problema sin Residual Block:

- En redes muy profundas, los gradientes se hacen cada vez más pequeños.
- Las capas iniciales casi no aprenden nada
- La red no mejora al añadir más capas

Solución con conexión residual:

- Si las capas convolucionales no aprenden nada útil: $F(x) \approx 0$
- Entonces: SALIDA = $F(x) + x = 0 + x = x$
- La red puede "saltar" capas que no son útiles
- Los gradientes fluyen directamente a través del atajo



Tokenización -> convierte palabras en números.

Ejemplo: ["This", "movie", "is", "great", "!"] → [2023, 3185, 2003, 1123, 999]

Embedding-> Cada palabra se representa como un punto en un espacio de 256 dimensiones. Palabras similares están cerca en el espacio.

Ejemplo: "movie" → [0.1, -0.3, 0.5, ..., 0.2] (256 valores)

"film" → [0.12, -0.28, 0.48, ..., 0.19] (similar a "movie")

Positional Encoding-> Añade información sobre la posición de cada palabra.

ENCODER LAYER 1...4

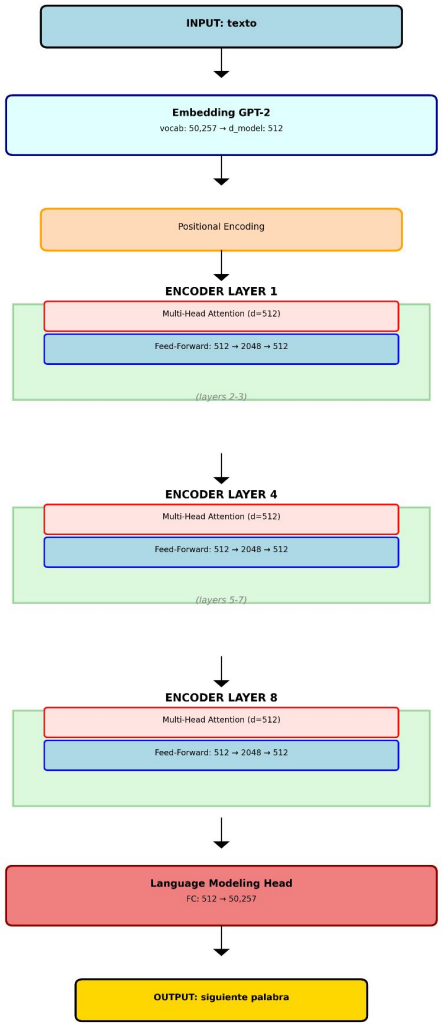
- Multi-Head Attention (8 heads)
- Feed-Forward: 256 → 1024 → 256

Cada encoder procesa el texto y aprende relaciones entre palabras.**Capa 1**: Relaciones sintácticas básicas ("This" → "movie" (sujeto-objeto)). **Capa 2:** Relaciones semánticas simples ("great" → palabras positivas relacionadas). **Capa 3:** Contexto más amplio (Relaciona "movie" con "great" aunque estén separadas). **Capa 4:** Representación final (Integra toda la información para clasificar)

Token [CLS]-> un token especial al inicio que representa toda la frase.

FC: 256 → 128 → 2 ReLU + Dropout (reduce dimensiones y clasifica en 2 categorías)

SALIDA: Positivo/Negativo Ejemplo: [0.05, 0.95] → 95% Positivo



Embedding GPT-2 -> Vocabulario: 50,257 tokens. dividido en d_model: 512 dimensiones.

Divide palabras en subpalabras, y permite manejar palabras nuevas que no están en el vocabulario

"running" → ["run", "##ning"]

Positional Encoding-> Añade información sobre la posición de cada palabra.

ENCODER LAYER 1...8

- Multi-Head Attention (8 cabezas, d=512)
- Feed-Forward: 512 → 2048 → 512

Language Modeling Head FC: 512 → 50,257 .Predice probabilidad para cada palabra del vocabulario.

Salida-> Predicción de siguiente palabra

Que es language Modeling-> La tarea de predecir la siguiente palabra dado un contexto. Lee texto palabra por palabra y predice la siguiente palabra en cada posición.

Perplexity-> La perplexity mide como de confundido está el modelo.

Mi resultado-> PPL = 562.73

- El modelo está "confundido" entre aproximadamente 563 palabras
- Es como si tuviera que adivinar entre 563 opciones igualmente probables

Menor perplexity= mejor modelo

RESULTADOS - COMPARACIÓN DE MÉTODOS

Experimento	DP Speedup	DDP Speedup	Ganador
MNIST + MLP	1.14x	1.18x	DDP
MNIST + CNN Small	1.04x	1.04x	Empate
CIFAR-10 + CNN Medium	1.05x	1.09x	DDP
CIFAR-100 + CNN Large	1.70x	1.78x	DDP
IMDB + Transformer Small	1.05x	1.07x	DDP
WikiText + Transformer Medium	1.11x	1.02x	DP

DDP mejor en 4/6 casos (67%)

DDP: Mejor en modelos grandes, peor con batch pequeños

DataParallel: Más simple, speedups modestos

CIFAR-100: Accuracy cae 6.3% (necesita ajuste de LR)

HALLAZGOS CLAVE

Expectativa 1: "Impacto de la paralelización de datos y modelos"**:**

Data parallelism (DP/DDP) funciona bien en visión

Speedup según Complejidad del modelo

CIFAR-100 (20M params): 1.78x speedup

Expectativa 2: "Relación entre arquitectura, tamaño y eficiencia"

Batch pequeño (WikiText bs=8) limita beneficio de DDP

Batch grande (MNIST bs=64) permite mejor speedup

Arquitecturas profundas (ResNet) se benefician más

Expectativa 3: "Estrategias trasladables a entornos multi-GPU reales":

DDP con 2+ GPUs reales debería lograr ~N×speedup

Learning rate debe escalarse: LR_nuevo = LR_base × √(N_GPUs)

ZeRO optimization para modelos >100M parámetros

LIMITACIONES Y QUE ME HA FALTADO

Probar con múltiples GPUs físicas

Razón: Hardware disponible (solo 1 GPU en Colab)

Impacto: Los speedups son más modestos de lo que serían con 2-4 GPUs reales

Mixed precision training no implementado:

He priorizado la comparación de métodos

Degradación de accuracy en CIFAR-100

Problema: Accuracy cae de 64.7% → 58.4% con DDP

Causa: Learning rate no ajustado para entrenamiento paralelo

Solución propuesta: Implementar linear warmup + LR scaling