Deep Learning a gran escala

César Olivares

Pontificia Universidad Católica del Perú
Maestría en Informática
INF659 - Técnicas avanzadas de data mining y sistemas inteligentes

2017

Aumento en el tamaño de los conjuntos de datos

- 1950s: Los primeros experimentos con redes neuronales artificiales.
- 1990s: Primeros éxitos comerciales de redes neuronales artificiales...
- ...pero era muy difícil entrenarlos y obtener un buen rendimiento.
- 2010s: Mientras más datos se tiene, es menos complicado obtener una buena generalización al entrenar modelos profundos.

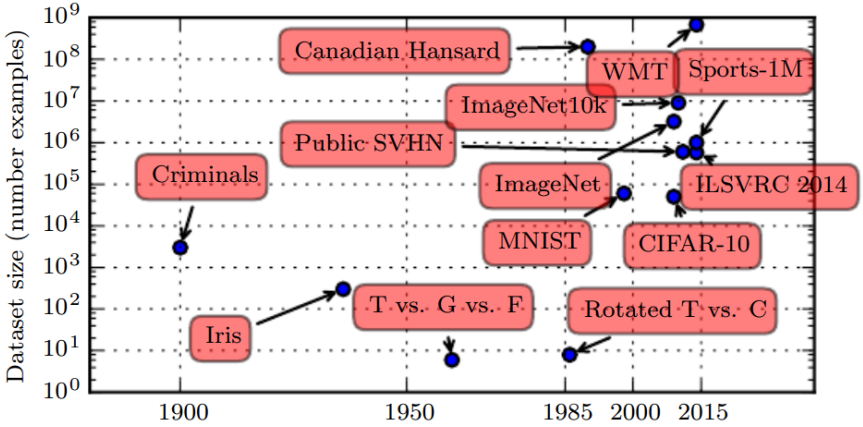

Figura 1: Tamaño de los conjuntos de datos a lo largo del tiempo, en escala logarítmica. (Goodfellow 2016)

Aumento en el número de conexiones por neurona

- Inicialmente el número de conexiones entre neuronas estaba limitado por el hardware.
- Hoy es en la práctica una consideración de diseño, y es común tener número de conexiones cercanos a las del cerebro de ratones o gatos.

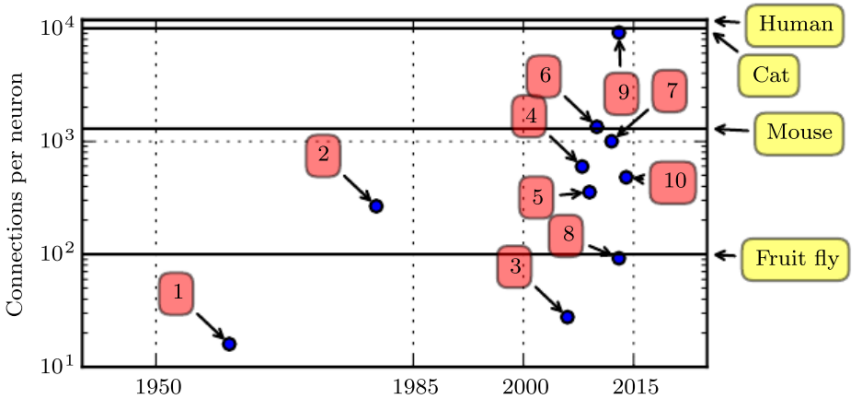

Figura 2: **Número de conexiones entre neuronas, en escala logarítmica (Goodfellow 2016).** 1. Adaptive linear element (Widrow 1960) 2. Neocognitron (Fukushima 1980) 3. GPU-accelerated convolutional network (Chellapilla 2006) 4. Deep Boltzmann machine (Salakhutdinov 2009) 5. Unsupervised convolutional network (Jarrett 2009) 6. GPU-accelerated multilayer perceptron (Ciresan 2010) 7. Distributed autoencoder (Le 2012) 8. Multi-GPU convolutional network (Krizhevsky 2012) 9. COTS HPC unsupervised convolutional network (Coates 2013) 10. GoogLeNet (Szegedy 2014)

Aumento en el número de unidades en las capas ocultas

- **Conexionismo:** una neurona en un modelo no es inteligente, pero una población *numerosa* de neuronas actuando juntas puede exhibir un comportamiento inteligente.
- Desde la introducción de unidades ocultas, las redes neuronales artificiales han duplicado su tamaño cada 2.4 años aproximadamente.

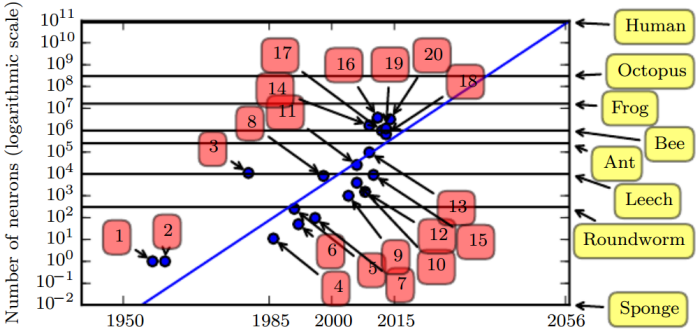

Figura 3: **Número de unidades ocultas, en escala logarítmica (Goodfellow 2016).** 1. Perceptron (Rosenblatt 1958) 2. Adaptive linear element (Widrow 1960) 3. Neocognitron (Fukushima 1980) 4. Early back-propagation network (Rumelhart 1986) 5. Recurrent neural network for speech recognition (Robinson 1991) 6. Multilayer perceptron for speech recognition (Bengio 1991) 7. Mean field sigmoid belief network (Saul 1996) 8. LeNet-5 (LeCun 1998) 9. Echo state network (Jaeger 2004) 10. Deep belief network (Hinton 2006) 11. GPU-accelerated convolutional network (Chellapilla 2006) 12. Deep Boltzmann machine (Salakhutdinov 2009) 13. GPU-accelerated deep belief network (Raina 2009) 14. Unsupervised convolutional network (Jarrett 2009) 15. GPU-accelerated multilayer perceptron (Ciresan 2010) 16. OMP-1 network (Coates and Ng 2011) 17. Distributed autoencoder (Le 2012) 18. Multi-GPU convolutional network (Krizhevsky 2012) 19. COTS HPC unsupervised convolutional network (Coates 2013) 20. GoogLeNet (Szegedy 2014)

Implementaciones veloces

- CPU
  - Antes del uso de GPUs, se mejoraba la velocidad 3x usando aritmética de punto fijo (Vanhoucke 2011).
  - Otras estrategias incluyen la optimización de estructuras de datos para evitar *cache misses* y el uso de instrucciones vectorizadas.
- GPU
  - Alto grado de paralelismo
  - Acceso a memoria con amplio ancho de banda, sin cache
  - Requiere sincronizar las instrucciones ejecutadas por diferentes hilos, agrupados en *warps*.
  - Dada la complejidad de escribir código para GPUs, se usa librerías tales como Theano, TensorFlow o Torch, que permiten especificar los modelos en términos de operaciones de nivel más alto, como multiplicaciones o convoluciones de matrices.

Implementaciones distribuidas y paralelizadas

- Paralelismo inter-modelos
- Paralelismo intra-modelo
- Paralelismo de datos

Paralelismo inter-modelos

- También conocido como ajuste o exploración del espacio de hiperparámetros.
- Consiste en la ejecución en paralelo de varias versiones de un mismo modelo, con diferentes hiperparámetros.
- Cada dispositivo o cada máquina es completamente independiente de los demás.

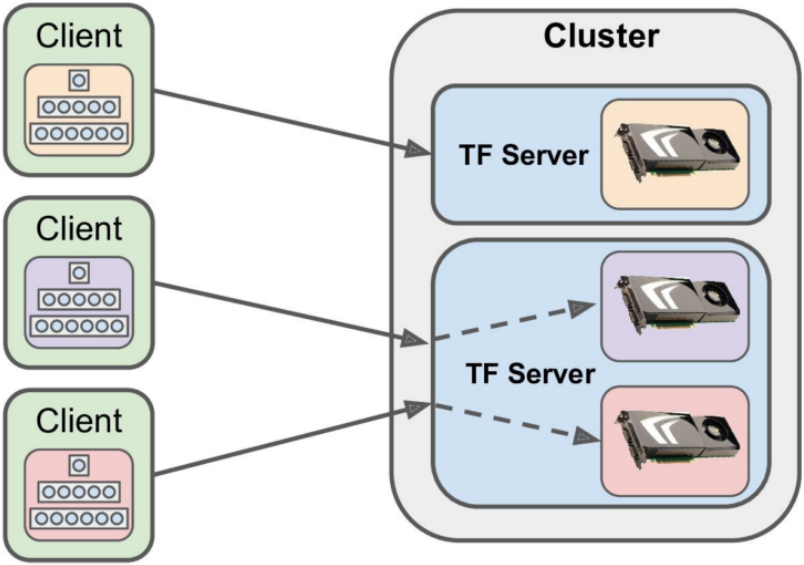

Figura 4: Entrenamiento de una red neuronal por dispositivo (Géron 2017).

Paralelismo inter-modelos: ensambles

- Se puede usar paralelismo inter-modelos para entrenar ensambles de modelos.
- Para la inferencia, se requerirá agregar las salidas o predicciones de los diversos modelos.
- Para coordinar la inferencia, se puede usar réplica intra-grafo o réplica inter-grafos.

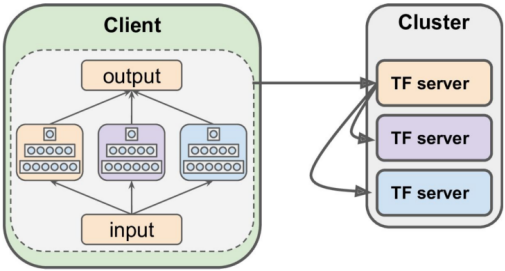

Figura 5: Réplica intra-grafo. Todo el ensamble es manejado como un único grafo. (Géron 2017)

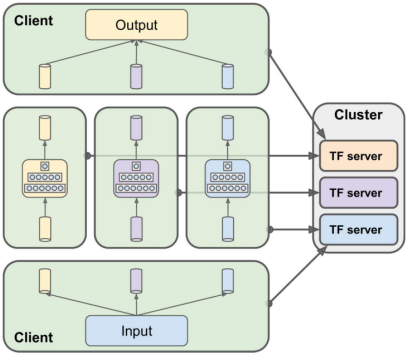

Figura 6: Réplica inter-grafos. Se requiere coordinar la sincronización entre los grafos. (Géron 2017)

Paralelismo intra-modelo: redes completamente conectadas

- Consiste en la ejecución en paralelo de diversas partes de un único modelo.
- Requiere decidir cómo dividir el modelo en partes y depende mucho de la arquitectura del modelo.
- No es muy efectivo para arquitecturas completamente conectadas. Si se distribuye las capas, cada capa tendrá que esperar la salida de la anterior. Si hace cortes verticales, requiere demasiada comunicación entre las partes (muy lento entre diversas máquinas).

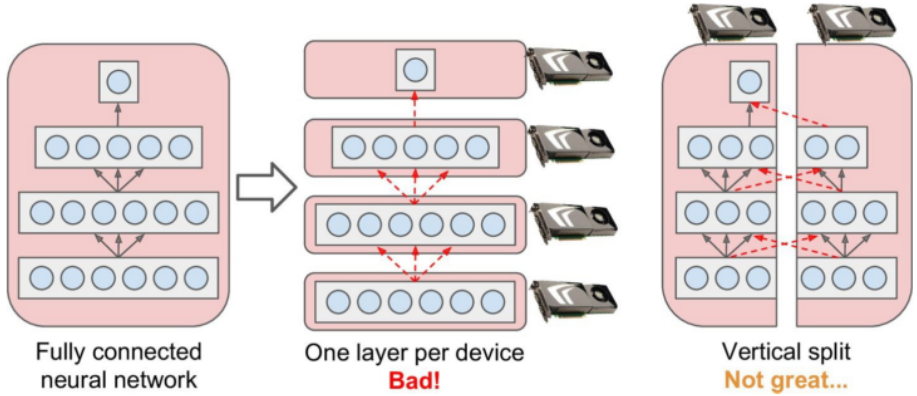

Figura 7: Paralelismo intra-modelo de una red completamente conectada (Géron 2017).

Paralelismo intra-modelo: redes parcialmente conectadas

- Algunas arquitecturas contienen capas sólo parcialmente conectadas a las anteriores. En estos casos es más fácil distribuir partes del modelo eficientemente.

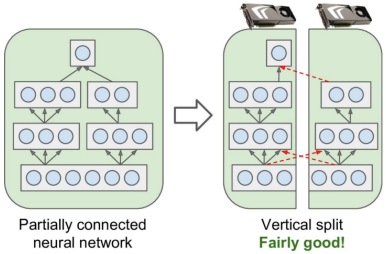

Figura 8: Paralelismo intra-modelo de una red parcialmente conectada. (Géron 2017)

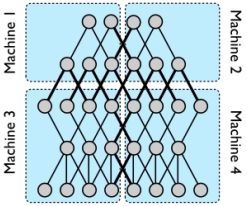

Figura 9: Un ejemplo de paralelismo intra-modelo usado en DistBelief. (Dean 2012)

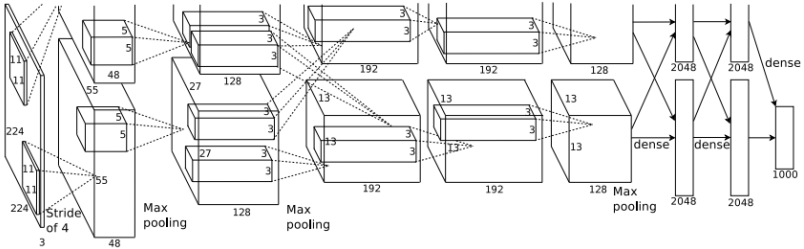

Figura 10: Paralelización con 2 GPUs en la arquitectura original de AlexNet. (Krizhevsky 2012)

Paralelismo intra-modelo: redes recurrentes

- Las arquitecturas recurrentes profundas se componen de varias capas de celdas. Dado que cada celda puede ser internamente compleja, el beneficio de ejecutar múltiples celdas en paralelo puede ser mayor que el costo de comunicación entre dispositivos.

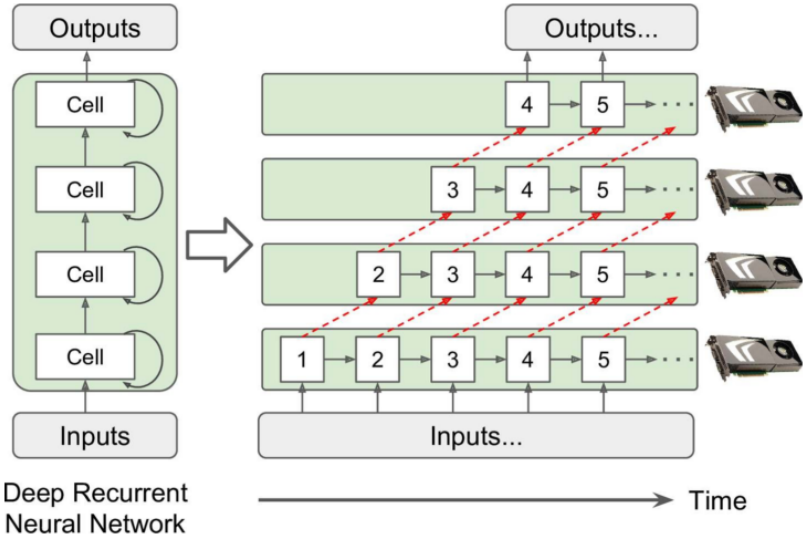

Figura 11: Paralelismo intra-modelo en una red recurrente profunda (Géron 2017).

Paralelismo de datos

- Consiste en la ejecución en paralelo de un único modelo, donde cada dispositivo o cada máquina procesa diversas partes del conjunto de datos, típicamente diferentes mini-batches.
- Como algoritmo de optimización se suele usar el **descenso de gradiente estocástico asíncrono** (Bengio 2001; Recht 2011).
- Los parámetros son actualizados en una memoria compartida o, cuando se usa diversas máquinas, en un servidor de parámetros (Dean 2012)
- El descenso de gradiente distribuido y asíncrono es en la actualidad el principal método de entrenamiento de redes profundas de gran escala (Chilimbi 2014; Wu 2015).

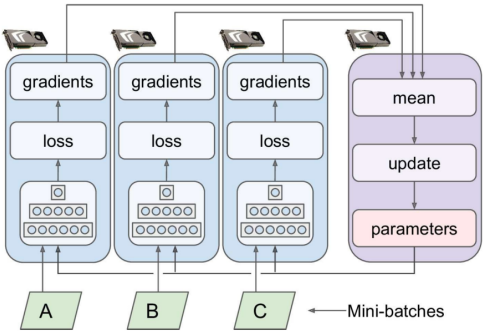

Figura 12: Paralelismo de datos (Géron 2017).

Paralelismo de datos / Saturación del ancho de banda

- Dada la necesidad de comunicarse con los servidores de parámetros al empezar y terminar cada paso del entrenamiento hay un límite a partir del cual añadir un GPU adicional no mejora el rendimiento, sino que aumenta la saturación de la comunicación y hace más lento el entrenamiento.
- Modelos pequeños con un conjunto de entrenamiento muy grande se entrenan mejor en una única máquina con un único GPU.
- Modelos grandes y densos saturan rápido pues requieren comunicar muchos parámetros.

Paralelismo de datos / Recomendaciones

- Agrupar los GPUs en pocos servidores para disminuir la comunicación a través de la red.
- Disgregar los parámetros en múltiples servidores de parámetros.
- Reducir la precisión flotante de los parámetros de 32 a 16 bits (p.ej.: tf.bfloat16 en lugar de tf.float32) disminuye a la mitad el tamaño de datos por transferir e impacta muy poco en el rendimiento del modelo.

Paralelismo de datos en TensorFlow

- En TensorFlow, cada *cluster* de servidores se compone de uno o más procesos servidores, denominados *tasks*, típicamente en diferentes equipos.
- Cada *task* pertenece a un *job*. Un *job* es un conjunto de *tasks* que juegan un mismo rol.
- Los principales *jobs* son *parameter server* (llevar cuenta del valor de los parámetros) y *worker* (realizar los cálculos del modelo).

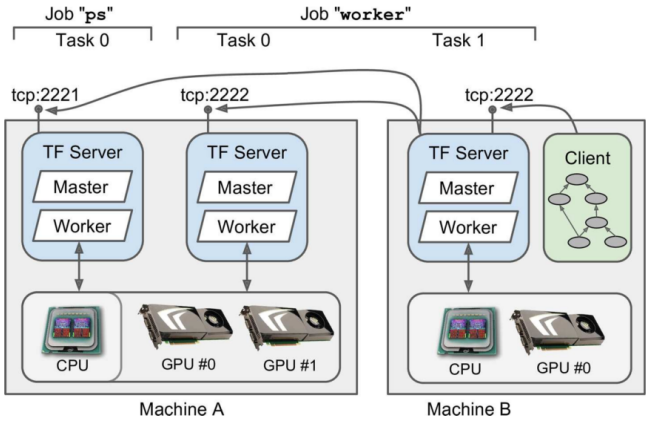

Figura 13: Cluster de TensorFlow (Géron 2017).

Paralelismo de datos en TensorFlow (2)

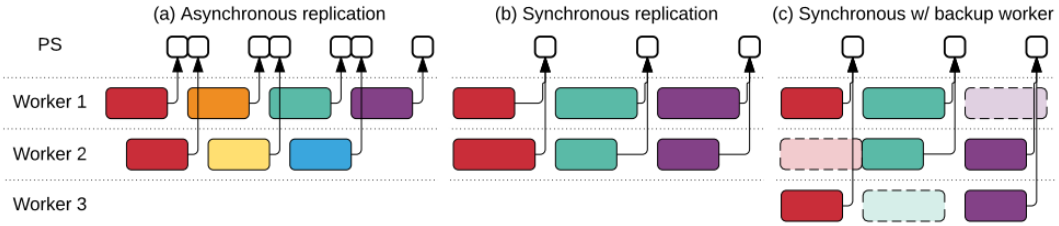

Figura 14: Tres esquemas de sincronización para SGD paralelo. Cada color representa el valor de inicio de un parámetro. Cada cuadrado en blanco es una actualización de parámetros. En (c), cada rectángulo rayado representa un *worker* cuyo resultado es desechado (Abadi 2016).

Modelo de ejecución de TensorFlow

- Tensorflow usa un único grafo de flujo de datos para representar todos los cálculos y estados de un algoritmo, incluidas las operaciones matemáticas, los parámetros, su optimización y el pre-procesamiento de las entradas.

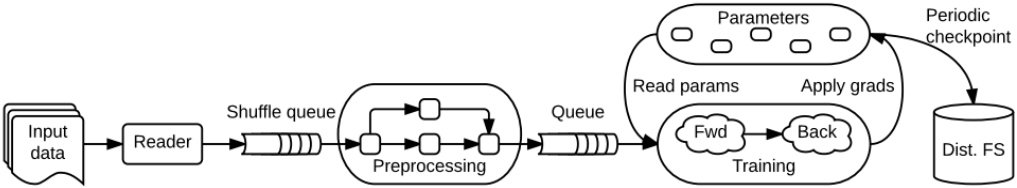

Figura 15: Grafo de flujo de datos en un *pipeline* de entrenamiento con TensorFlow, incluyendo subgrafos para la lectura de los datos de entrada, el preprocesamiento, el entrenamiento y la generación de *checkpoints* (Abadi 2016).

Compresión de modelos

- Los modelos grandes suelen obtener menor error.
  - Modelo grande entrenado con *dropout*
  - Ensamble de varios modelos
- A menudo se necesita modelos pequeños con bajo consumo de recursos.
- Una buena estrategia (Buciluǎ 2006) es entrenar un modelo pequeño para que imite al modelo grande, mediante el uso de datos sintéticos cercanos a los datos originales.
- Se obtiene menor error que entrenando directamente un modelo pequeño.

Bibliografía

- Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... others. (2016). TensorFlow: A System for Large-Scale Machine Learning. In OSDI (Vol. 16, pp. 265–283).
- Bengio, Y., Ducharme, R., & Vincent, P. (2001). A neural probabilistic language model. In T. K. Leen, T. G. Dietterich, & V. Tresp, editors, NIPS'2000 , pages 932–938. MIT Press.
- Buciluǎ, C., Caruana, R., & Niculescu-Mizil, A. (2006). Model compression. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 535–541).
- Chilimbi, T., Suzue, Y., Apacible, J., & Kalyanaraman, K. (2014). Project Adam: Building an efficient and scalable deep learning training system. In 11th USENIX Symposium on Operating Systems Design and Implementation (OSDI'14).
- Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Le, Q., Mao, M., Ranzato, M., Senior, A., Tucker, P., Yang, K., & Ng, A. Y. (2012). Large scale distributed deep networks. In NIPS'2012 .
- Géron, A. (2017). Hands-On Machine Learning with Scikit-Learn and TensorFlow Concepts, Tools, and Techniques to Build Intelligent Systems O'Reilly Media.
- Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. MIT Press. Retrieved from http://www.deeplearningbook.org/
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances In Neural Information Processing Systems, 1–9.
- Recht, B., Re, C., Wright, S., & Niu, F. (2011). Hogwild: A lock-free approach to parallelizing stochastic gradient descent. In NIPS'2011 .
- Vanhoucke, V., Senior, A., & Mao, M. Z. (2011). Improving the speed of neural networks on CPUs. In Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop.
- Wu, R., Yan, S., Shan, Y., Dang, Q. & Sun, G. (2015). Deep image: Scaling up image recognition. arXiv:1501.02876.