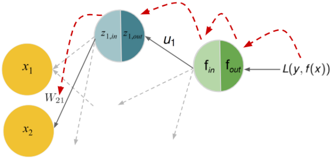
Deep Learning

Hardware and Software



Learning goals

-
-
-

Hardware for Deep Learning

HARDWARE FOR DEEP LEARNING

- Deep neural networks require special hardware to be trained efficiently.
- The training is done using **G**raphics **P**rocessing **U**nits (GPUs) and a special programming language called CUDA.
- Training on standard CPUs takes a very long time.

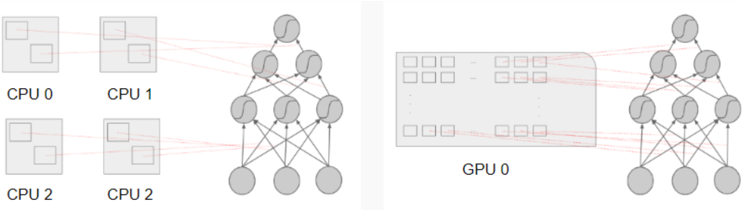


Figure: *Left:* Each CPU can do 2-8 parallel computations. *Right:* A single GPU can do thousands of simple parallel computations.

GRAPHICS PROCESSING UNITS (GPUS)

- Initially developed to accelerate the creation of graphics
- Massively parallel: identical and independent computations for every pixel
- Computer Graphics makes heavy use of linear algebra (just like neural networks)
- Less flexible than CPUs: all threads in a core concurrently execute the same instruction on different data.
- Very fast for CNNs, RNNs need more time
- Popular ones: GTX 1080 Ti, RTX 3080 / 2080 Ti, Titan RTX, Tesla V100 / A100
- Hundreds of threads per core, few thousands cores, around 10 teraFLOPS in single precision, some 10s GBs of memory
- Memory is important - some SOTA architectures do not fit GPUs with <10 GB

TENSOR PROCESSING UNITS (TPUS)

- Specialized and proprietary chip for deep learning developed by Google
- Hundreds of teraFLOPS per chip
- Can be connected together in *pods* of thousands TPUs each (result: hundreds of **peta**FLOPS per pod)
- Not a consumer product! Can be used in the Google Cloud Platform (from 1.35 USD / TPU / hour) or Google Colab (free!)
- Enables DeepMind to make impressive progress : AlphaZero for Chess became world champion after just 4 hours of training concurrently on 5064 TPUs

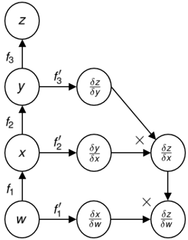
AND EVERYTHING ELSE...

- With such powerful devices, memory/disk access during training become the bottleneck
  - Nvidia DGX-1: Specialized solution with eight Tesla V100 GPUs, dual Intel Xeon, 512 GB of RAM, 4 SSD disks of 2TB each
- Specialized hardware for on-device inference
  - Example: Neural Engine on the Apple A11 (used for FaceID)
  - Keywords/buzzwords: *Edge computing* and *Federated learning*

Software for Deep Learning

SOFTWARE FOR DEEP LEARNING

- CUDA is a very *low level* programming language and thus writing code for deep learning requires a lot of work.
- Deep learning (software) frameworks:
  - Abstract the hardware (same code for CPU/GPU/TPU)
  - Automatically differentiate all computations
  - Distribute training among several hosts
  - Provide facilities for visualizing and debugging models
  - Can be used from several programming languages
  - Based on the concept of *computational graph*



SOFTWARE FOR DEEP LEARNING

Tensorflow

- Popular in the industry
- Developed by Google and open source community
- Python, R, C++ and Javascript APIs
- Distributed training on GPUs and TPUs
- Tools for visualizing neural nets, running them efficiently on phones and embedded devices.

Keras

- Intuitive, high-level **wrapper** on Tensorflow for rapid prototyping
- Python and (unofficial) R APIs

SOFTWARE FOR DEEP LEARNING

Pytorch

- Popular in academia
- Supported by Facebook
- Python and C++ APIs
- Distributed training on GPUs

MXNet

- Open-source deep learning framework written in C++ and cuda (used by Amazon for their Amazon Web Services)
- Scalable, allowing fast model training
- Supports flexible model programming and multiple languages (C++, Python, Julia, Matlab, JavaScript, Go, **R**, Scala, Perl)

Example: MNIST digit recognizer

MNIST DIGIT RECOGNIZER

- The MNIST dataset is a large dataset of handwritten digits (black and white) that is commonly used for benchmarking various image processing algorithms.
- It is a good dataset for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal effort on preprocessing and formatting.
- There have been a number of scientific papers on attempts to achieve the lowest error rate. One paper, using a hierarchical system of convolutional neural networks (chapter 5), manages to get an error rate of only 0.23 percent.

MNIST DIGIT RECOGNIZER

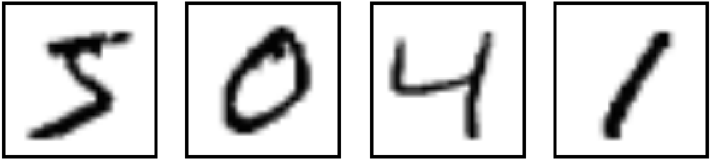
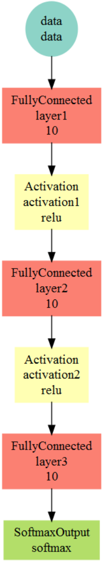


Figure: Snipped from the mnist data set (LeCun and Cortes (2010)).

- 70k image data of handwritten digits with 28×28 pixels.
- Classification task with 10 classes (0, 1, 2, ..., 9).

MNIST DIGIT RECOGNIZER

- We attempt classification with the model on the right:

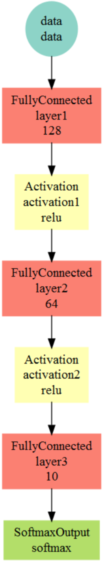


MNIST DIGIT RECOGNIZER

- We used SGD with a minibatch of size 100 and trained for 10 epochs.
- Consequently we feed our algorithm successively with 100 training samples before updating the weights.
- After 10 epochs, our neural network begins to stagnate at a training accuracy of roughly 93.5%
- Next, we use the model to predict the test data.
- We find that the accuracy of the model on the test data is only 89.843% which is unsatisfactory.

MNIST DIGIT RECOGNIZER

- Because the performance of the previous model was somewhat poor, we try the following, much larger, network (all other parameters remain the same)
- Rerunning the training with the new architecture, this model yields us a training accuracy of 99.39% and a test accuracy of 96.514%.



KEY HYPERPARAMETERS

- In addition to the structure/topology of the neural network, the performance of a model is also strongly affected by some key hyperparameters such as:
  - α, the learning rate
  - λ, the regularization coefficient
  - T, the number of training iterations
  - m, the minibatch size
  - and others...
- These hyperparameters typically control the complexity of the model and the convergence of the training algorithm.
- In the next couple of lectures, we'll examine methods and techniques to set these hyperparameters and the theoretical motivations behind many of them.

REFERENCES

Yann LeCun and Corinna Cortes (2010)
MNIST handwritten digit database
http://yann.lecun.com/exdb/mnist/