ISC High Performance IMAGINE TOMORROW

Tutorial. May 12, 2024 2-6 pm

Distributed Training of Deep Neural Networks

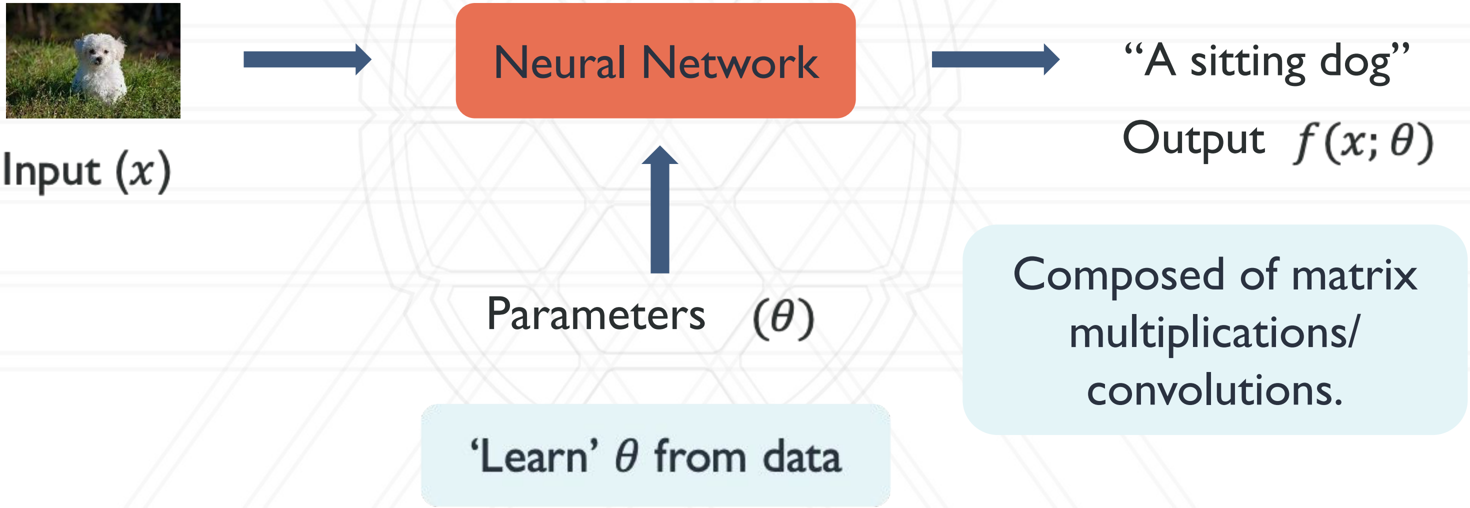
Abhinav Bhatele, Siddharth Singh, Daniel Nichols
Department of Computer Science

PARALLEL SOFTWARE AND SYSTEMS GROUP

UNIVERSITY OF MARYLAND

Neural Networks

- Neural Networks (NNs): ‘Parameterized’ function approximators
- Can work with very high dimensional data.



Input (x)

Neural Network

“A sitting dog”

Output $f(x;\theta)$

Parameters (θ)

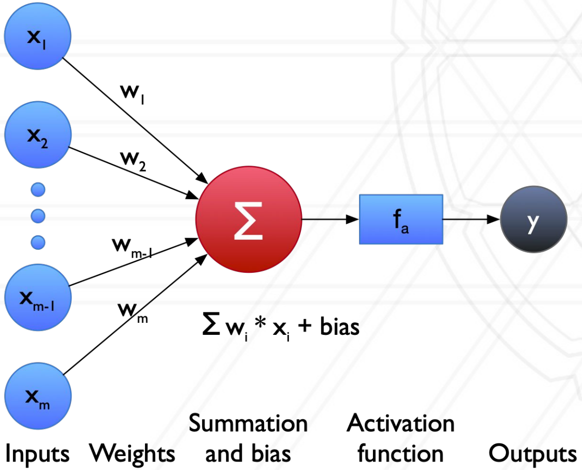
Composed of matrix multiplications/ convolutions.

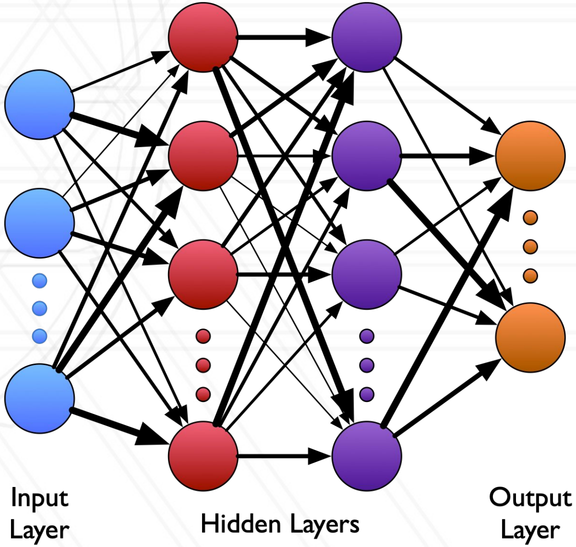
‘Learn’ θ from data

Deep neural networks

Neural networks can be used to model complex functions

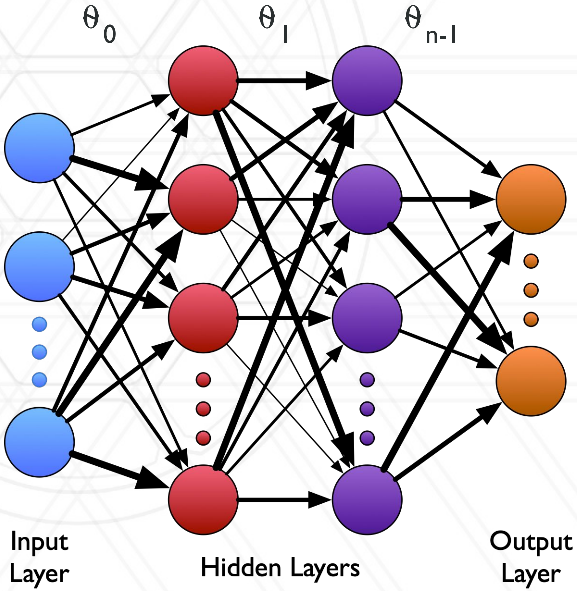
Several layers that process input data





Training a neural network

Problem: Find a set of weights/parameters that best fits the function we are trying to learn over a given training dataset



Other terms and definitions

- Loss: a scalar proxy that when minimized leads to higher accuracy
- Learning/training: task of selecting weights that lead to an accurate function / minimizes the loss
- Gradient descent: process of updating the weights using gradients (derivatives) of the loss weighted by a learning rate
- Batch: Small subsets of the dataset processed independently
- Epoch: One pass over all the batches

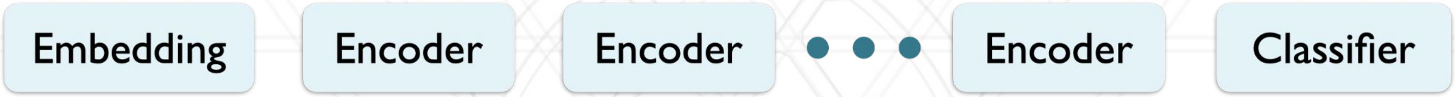
Stochastic Gradient Descent

Divide training data into batches

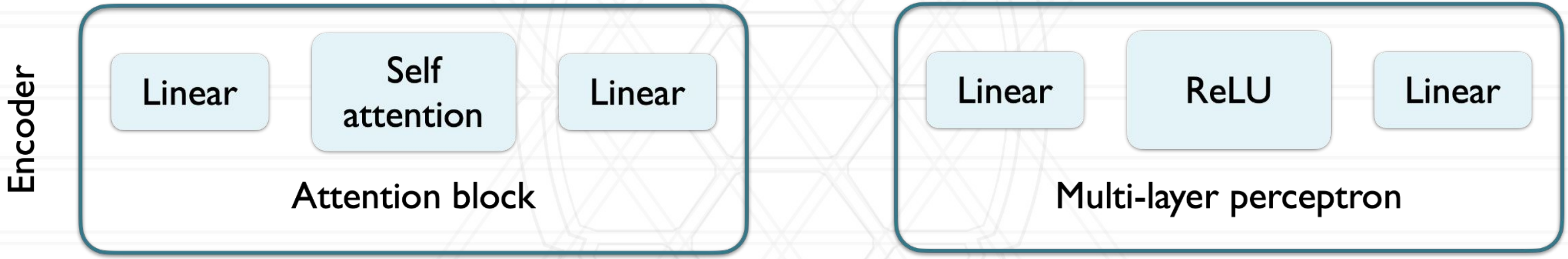
Repeat the following steps until loss, L, is minimized sufficiently:

- Read in one batch of training data
- Forward pass: Compute the activation, $f(x;\theta)$, and loss, L, on the batch
- Backward: Calculate gradients of the loss w.r.t. the parameters via backpropagation $\frac{\partial L}{\partial \theta}$
- Optimizer step: Use gradients to update weights/parameters, θ, such that loss is incrementally reduced

Where are the matrix multiplies?



Embedding → Encoder → Encoder • • • Encoder → Classifier



Encoder

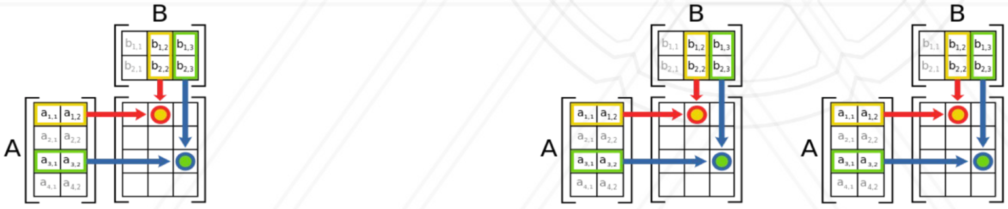
Linear, Self attention, Linear

Attention block

Linear, ReLU, Linear

Multi-layer perceptron

Linear



Get the tutorial repository

- Clone the git repository as follows:

```
git clone https://github.com/axonn-ai/distrib-dl-tutorial.git
```

PyTorch

- torch – a Python library for tensor computations with GPU support
- torch.nn – library for training deep neural networks

- We will start with looking at single GPU training using PyTorch

Training task

- Image classification using MNIST data

Using PyTorch

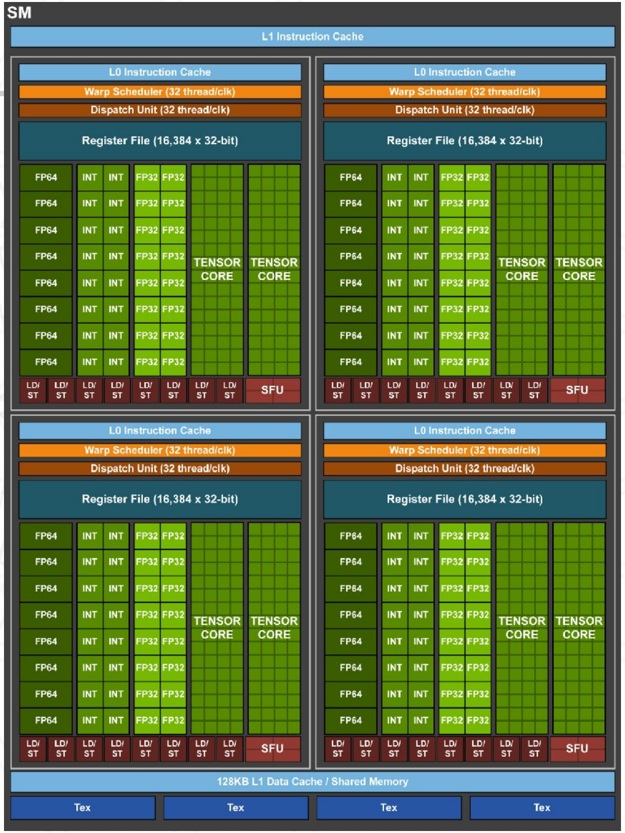
- Code location in the tutorial repo: session_1_basics/train.py

```
$ cd session_1_basics/
$ sbatch --reservation=isc2024 run.sh
```

Parameter	
--num-layers	4
--hidden-size	2048
--image-size	64
--data-dir	<path-to-data>
--batch-size	32
--lr	0.001

Mixed-precision Training

- GPUs have FP32, FP64 and tensor cores
- We can optimize performance by doing some operations in lower precision


SM
L1 Instruction Cache
L0 Instruction Cache
Warp Scheduler (32 thread/clk)
Dispatch Unit (32 thread/clk)
Register File (16,384 x 32-bit)
FP64
INT
FP32
TENSOR CORE
LD/ ST
SFU
128KB L1 Data Cache / Shared Memory
Tex

Mixed-precision Training

- Code location in the tutorial repo: session_1_basics/train_mp.py

```
MIXED_PRECISION=true sbatch --reservation=isc2024 run.sh
```

Activation Checkpointing

- Activations are outputs of individual layers
- To save memory, we checkpoint only inputs to each layer
  - Regenerate intermediate and output activations as needed in the backward pass
- Code location in the tutorial repo: session_1_basics/train_mp.py

```
CHECKPOINT_ACTIVATIONS=true sbatch --reservation=isc2024 run.sh
```

Abhinav Bhatele and Siddharth Singh

Department of Computer Science

bhatele@umd.edu, ssingh37@umd.edu