Neural networks

- Neural networks are made up of **nodes** or **units**, connected by **links**
- Each link has an associated **weight** and **activation level**
- Each node has an **input function** (typically summing over weighted inputs), an **activation function**, and an **output**
Neuron Model: Logistic Unit

“bias unit”

\[ h_\theta(x) = g(\theta^T x) \]
\[ = \frac{1}{1 + e^{-\theta^T x}} \]

Sigmoid (logistic) activation function:
\[ g(z) = \frac{1}{1 + e^{-z}} \]

Based on slide by Andrew Ng
Neural Network

\[
h_\theta(x) = \sigma(x) = \sigma(Wx + b)
\]

Layer 1 (Input Layer)
Layer 2 (Hidden Layer)
Layer 3 (Output Layer)

bias units

\(x_0\)

\(x_1\)

\(x_2\)

\(x_3\)

\(a_0^{(2)}\)

\(a_1^{(2)}\)

\(a_2^{(2)}\)

\(a_3^{(2)}\)
Neural Network Training

- Feedforward and back propagation gradient descend

source: https://www.programmersought.com/article/45312377380/
Interlude: Why we need gradient?

Machine learning is basically an optimization process.

One optimization algorithm is gradient descent.

DNN training use a variant called stochastic gradient descent (SGD).

\[ \theta = \theta - \alpha \nabla f(x; \theta) \]

\( \theta \) represents all the weight as a vector.

\( \alpha \) is the learning rate, hyperparameter.

\( \nabla f(x; \theta) \) is the gradient.

https://builtin.com/data-science/gradient-descent
A simple (sort of real) example

Image classification (2 class, 0 and 1) with 3x3 binary image

Model design:
1. 2x2 kernel Convolution layer
2. ReLU activation layer (max function)
3. 4x1 linear layer
4. Sigmoid function

Cost function:
Square error (l2 norm)

\[ E = (y - \hat{y})^2 \]

\[ A = [a_1 \ldots a_4] \]
\[ X = [x_1 \ldots x_4] \]
\[ y = a_1x_1 + a_2x_2 + a_3x_3 + a_4x_4 \]

Gif from https://medium.datadriveninvestor.com/introduction-to-how-cnns-work-77e0e4cde99b
A simple (sort of real) example

All blue variables are our learning target
We called it weight (parameter)
* Different from hyperparameter like learning rate which is adjusted by human being
A simple (sort of real) example

Image 0  Conv kernel

$A = [3, 2, 1, 0]$
Batch Gradient Descend

source: https://www.baeldung.com/cs/epoch-vs-batch-vs-mini-batch
Stochastic Gradient Descend

source: https://www.baeldung.com/cs/epoch-vs-batch-vs-mini-batch
Minibatch-Based SGD

Training Data

$S_1$

$S_2$

$S_3$

$S_4$

$\ldots$

$S_{n-1}$

$S_n$

Iteration 1

Iteration 2

Iteration n/2

Compute Gradients

Update Weights

source: https://www.baeldung.com/cs/epoch-vs-batch-vs-mini-batch
Lecture 13
Distributed Training

Song Han
songhan@mit.edu
Models are getting larger and larger

Better model always comes with higher computational cost (vision)

Figures from Once-for-all project page.
Models are getting larger and larger

Better model always comes with higher computational cost (NLP)

NLP model size is increasing exponentially

<table>
<thead>
<tr>
<th>Year</th>
<th>Model Size (#Params in Billion)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2017</td>
<td>Google Transformer 0.05B</td>
</tr>
<tr>
<td>2018</td>
<td>OpenAI GPT 0.11B</td>
</tr>
<tr>
<td></td>
<td>Google BERT 0.34B</td>
</tr>
<tr>
<td>2020</td>
<td>OpenAI GPT-2 1.5B</td>
</tr>
<tr>
<td>2021</td>
<td>MegatronLM 8.3B</td>
</tr>
<tr>
<td></td>
<td>T-NLG 17B</td>
</tr>
</tbody>
</table>

*Measured on Nvidia A100

Figures from Microsoft Turing Project

175 Billion model parameters
8 Million web pages
3 Million GPU hours*

*Measured on Nvidia A100
Models are getting larger and larger

Large Models Take Longer Time to Train

<table>
<thead>
<tr>
<th>Models</th>
<th>#Params (M)</th>
<th>Training Time (GPU Hours)</th>
<th>Measured on Nvidia A100</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-50</td>
<td>26</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td>ResNet-101</td>
<td>45</td>
<td>44</td>
<td></td>
</tr>
<tr>
<td>BERT-Base</td>
<td>108</td>
<td>84</td>
<td></td>
</tr>
<tr>
<td>Turing-NLG 17B</td>
<td>17,000</td>
<td>TBA</td>
<td></td>
</tr>
<tr>
<td>GPT-3 175B</td>
<td>175,000</td>
<td>3,100,000</td>
<td></td>
</tr>
</tbody>
</table>

If without distributed training, a single GPU would take **335 years** to finish GPT-3!
Distributed Training is Necessary

- Developers / Researchers’ time are more valuable than hardware.
- If a training takes 10 GPU days
  - Parallelize with distributed training
  - 1024 GPUs can finish in 14 minutes (ideally)!
- The develop and research cycle will be greatly boosted

Let’s see a use case of distributed training!
Parallelism in Distributed Training

- Data Parallelism
- Model Parallelism
- Compare the Advantages and Disadvantages of Two Parallelism
Introduction to Distributed Training

Data Parallelism

ML Model

Training Dataset

Data Parallelism

GPU 1

GPU 2

...
Introduction to Distributed Training

Data Parallelism

ML Model

Split the data

Training Dataset

GPU 1

GPU 2

... 

GPU N
**Introduction to Distributed Training**

**Data Parallelism**

- **ML Model**
- **Training Dataset**
- **GPUs**: GPU 1, GPU 2, ..., GPU N

- Same model across devices
Dive into Data Parallelism
Scaling Distributed Machine Learning with the Parameter Server

Parameter Server
The central controller of the whole training process

Two different roles in framework:
- **Parameter Server**: receive gradients from workers and send back the aggregated results
- **Workers**: compute gradients using splitted dataset and send to parameter server

Worker nodes
The hardware accelerators and dataset storage.

Scaling Distributed Machine Learning with the Parameter Server. Mu Li et al. 2014
Figure credits from: Deep Gradient Compression. Lin et al. 2018
Problems with Parameter Server

When number of workers is small 😊

When number of workers is large 😞

The bandwidth requirement of parameter server grows linearly w.r.t number of workers.
Distributed Communication

Point-to-Point: Send and Recv

Send: n0 -> n3

Recv: n0 -> n3

Point-to-point communication: transfer data from one process to another

- Send & Receive are the most common distributed communication schemes.
- Implemented in Socket / MPI / Gloo / NCCL
Distributed Communication
Naive All-Reduce Implementation - Parallel Reduce

Pseudocode:

```
Parallel for i:=0 to N:
   Allreduce(work[i])
```

Perform ALL reduce operations simultaneously.

Time: $O(1) \leftarrow$ improved

Bandwidth: $O(N^2) \leftarrow$ worse
Recursive Halving All Reduce

**Step 1** - Each node exchanges with neighbors with offset 1

**Step 2** - Each node exchanges with neighbors with offset 2

**Step 3** - Each node exchanges with neighbors with offset 4

For N workers, AllReduce finish in $\log(N)$ steps.

Distributed Communication

All Reduce Implementations Comparison

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
<th>Peak Node Bandwidth</th>
<th>Total Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parameter Server</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>All-Reduce - Sequential</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>All-Reduce - Ring</td>
<td>$O(N)$</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>All-Reduce - Parallel</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>All-Reduce - Recursive Halving</td>
<td>$O(lgN)$</td>
<td>$O(1)$</td>
<td>$O(N)$</td>
</tr>
</tbody>
</table>

AllReduce with proper implementations reduce the peak bandwidth from $O(N)$ to $O(1)$ with little time overhead.
Parallelism in Distributed Training

• Data Parallelism
• Model Parallelism
• Compare the Advantages and Disadvantages of Two Parallelism
Data Parallelism Cannot Train Large Models

Though model parallelism has better device utilization, if train a super-large model (e.g., GPT-3)

Even the best GPU CANNOT fit the model into memory!
Introduction to Distributed Training

Model Parallelism

Figures credit from CMU 15-849 [Jia 2022]
In order to fit training into hardware, instead of splitting the data, model parallelism split the model.

350GB / 8 cards = 43.75G < 80G

With model parallelism, large ML models can be placed and trained on GPUs.
Introduction to Distributed Training

Model Parallelism

ML Model

Training Dataset

Single copy of data

Figures credit from CMU 15-849 [Jia 2022]
Introduction to Distributed Training

Model Parallelism

Split the model

Figures credit from CMU 15-849 [Jia 2022]
Introduction to Distributed Training

Model Parallelism

Figures credit from CMU 15-849 [Jia 2022]
Introduction to Distributed Training

Model Parallelism

Training Dataset

ML Model

Split the model

Figures credit from CMU 15-849 [Jia 2022]
Model Parallelism Workflow

Naive Implementation

(a). Training data flow

(b). Training timeline

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]
Model Parallelism Workflow

Naive Implementation

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
Model Parallelism Workflow

Naive Implementation

(a). Training data flow

(b). Training timeline

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
Model Parallelism Workflow

Naive Implementation

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
Model Parallelism Workflow

Naive Implementation

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
Model Parallelism Workflow

Naive Implementation

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.
Model Parallelism Workflow

Naive Implementation

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
Model Parallelism Workflow

Naive Implementation

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

Figures from GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism
Model Parallelism Workflow

Naive Implementation

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.
Model Parallelism Workflow

(a). Training data flow

F: Forward  B: Backward. Train a 4 layer network with model parallelism.

Model parallelism is needed for training a bigger DNN model on accelerators by dividing the model into partitions and assigning different partitions to different accelerators.

But accelerators are significantly under-utilized during model parallelism!

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]
Naive Model Parallelism Suffers from Utilization

Only one device is computing at a time and others are waiting for it.

Theoretical utilization: 25% (low!)

Usual data parallelism utilization: ~75%

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]
**Pipeline Parallelism**

**Gpipe: Easy Scaling with Micro-Batch Pipeline Parallelism**

- Split a single batch to micro batches
  - [16, 10, 512] ->
    - [4, 10, 512]
    - [4, 10, 512]
    - [4, 10, 512]
  - Motivation: model parameters are not changed during computation within a batch, thus we can pipeline computation and communication

---

(a). Naïve model parallelism

(b). GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]
Pipeline Parallelism

Micro-batch improves the device utilization

(a). Naive model parallelism

(b). Pipeline parallelism

GPipe: Efficient Training of Giant Neural Networks using Pipeline Parallelism [Huang et al. 2018]
Parallelism in Distributed Training

• Data Parallelism
• Model Parallelism
• Compare the Advantages and Disadvantages of Two Parallelism
Comparison between two parallelism

**Data Parallelism:**
- Split the data
- Same model across devices
- Easy to parallelize, high utilization
- N copies of model

**Model Parallelism:**
- Split the model
- Move activations through devices
- Hard to parallelize, load balancing issue
- Single copy of model

Figures credit from CMU 15-849 [Jia 2022]
Other Systems for ML Topics

• Systems for other types of ML/AI tasks (e.g., reinforcement, LLM, etc.)
• Memory-efficient training
• Efficient hyperparameter and model architecture search
• Efficient inference and model serving
• Security in ML
• Edge ML (e.g., federated learning)
Machine Learning for Systems

- ML for compiler and program analysis/generation/debugging
- ML for resource allocation
- ML for scheduling
- ML for memory management
- ML for security
- ML for hardware design