Sign up for SRC session and get an open AI account for upcoming SRC assignment

CSCI 1470/2470
Spring 2023

Ritambhara Singh

February 10, 2023
Friday

DALL-E 2 prompt “a painting of deep underwater with a yellow submarine in the bottom right corner”
Today’s goal – learn about role of matrices and introduction to Tensorflow framework

(1) How matrix operations make learning efficient

(2) Batching and broadcasting
   (It’s all about matching dimensions in matrix operations!)

(3) Intro to Tensorflow
Recap: Simple Neural net (w/ linear unit)

\[ \text{logit} = w_1 x_1 + w_2 x_2 + \cdots + w_k x_k + b \]
Matrix-multiplication Style Neural Net

What are the dimensions? (keeping bias separate)

\[
\text{logit} = w_1x_1 + w_2x_2 + \cdots + w_kx_k + b
\]

• Really, this is just

\[
\text{logit} = Wx + b
\]

• \(W\) = vector of weights (1 × 7 in this example)
• \(x\) = input as a vector (7 × 1 in this example)
• \(b\) = scalar bias (1 × 1)
Fully connected layer with multiple outputs

$m$ outputs means
- $m$ sets of linear functions

which means:
- $m$ sets of weight vectors (or a weight matrix)
- $m$ biases

What are the dimensions?
Fully connected layer with multiple outputs

\[ \text{logit} = Wx + b \]

- \( W \) = matrix of weights (3×7 in this example)
- \( x \) = input as a vector (7×1 in this example)
- \( b \) = vector bias (3×1)
Fully connected layer with multiple outputs

- dimensions of $\mathbf{W} = (m, n)$
- Dimensions of $\mathbf{b} = (m, 1)$
- $\mathbf{Wx} + \mathbf{b}$ then is a $(m, n) \times (n, 1) + (m, 1)$
Gradient Updates using Matrices

• Previously: $\Delta w_{i,j} = -\alpha \cdot \frac{\partial L}{\partial w_{i,j}}$

• With Matrices: $\Delta W = -\alpha \cdot \nabla_w L$

10x784 matrix of weights

Jacobian matrix: matrix of all first-order partial derivatives for a vector-valued function.

10x784 matrix of partial derivatives of loss w.r.t. weights

i.e. $\left[ \begin{array}{ccc} \frac{\partial L}{\partial w_{1,1}} & \frac{\partial L}{\partial w_{1,2}} & \cdots & \frac{\partial L}{\partial w_{1,784}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial L}{\partial w_{10,1}} & \frac{\partial L}{\partial w_{10,2}} & \cdots & \frac{\partial L}{\partial w_{10,784}} \end{array} \right]$
Why is matrix formulation useful?

Remember the three loops in last lecture?
Existing linear algebra optimizations

- Matrix multiplication can be **way** faster than *for* loops
- Example: time required to compute dot product of $a, b \in \mathbb{R}^{1,000,000}$

![Comparison between for loops and matrix multiplication](https://www.coursera.org/lecture/neural-networks-deep-learning/vectorization-NYnog)

- Lots of existing effort to build fast linear algebra code (e.g. NumPy)
- Leads to order of magnitude speedup!
GPUs to the rescue!

• *Graphics* Processing Units

• GPUs are really good at computing mathematical operations in parallel!

• Matrix multiplication == many *independent* multiply and add operations

  Easily parallelizable

  GPUs are great for this!

Image courtesy: https://global.aorus.com/blog-detail.php?id=878
CPU v/s GPU

output

Write back

ALU

Arithmetic logic unit

Decode

Fetch

input
CPU v/s GPU

Vector operations (SSE / AVX)

GPU: specialized accelerator

- Fetch
- Decode
- ALU
- Write back
GPU-Parallel Acceleration

• User code (*kernels*) is compiled on the *host* (the CPU) and then transferred to the *device* (the GPU)

• Kernel is executed as a *grid*

• Each grid has multiple *thread blocks*

• Each thread block has multiple *warps*

A warp is the basic schedule unit in kernel execution

A warp consists of 32 threads

 CUDA compute model

Compute Unified Device Architecture is a parallel computing platform and application programming interface (API)
GPU-Parallel Acceleration

- Programmer decides how they want to parallelize the computation across grids and blocks
  - Modern deep learning frameworks take care of this for you

- CUDA compiler figures out how to schedule these units of computation on to the physical hardware

https://www.researchgate.net/publication/236666656_Accelerating_Fibre_Orientation_Estimation_from_Diffusion_Weighted_Magnetic_Resonance_Imaging_Using_GPUs
GPU-Parallel Acceleration

- Upshot: order of magnitude speedups!
- Example: training CNN on CIFAR-10 dataset

CUDA compute model

https://www.researchgate.net/publication/236666656_Accelerating_Fibre_Orientation_Estimation_from_Diffusion_Weighted_Magnetic_Resonance_Imaging_Using_GPUs

https://medium.com/@andriylazorenko/tensorflow-performance-test-cpu-vs-gpu-79fcd39170c

Any questions?
Batching and broadcasting
Computing a “batch” of outputs

• We can compute output of a single n x 1 input by multiplying it by weight matrix

\[ \begin{align*}
W \text{ (dims: m by n)} & \quad X \text{ (dims: n by 1)} & \quad \text{output (dims: m by 1)} \\
\begin{array}{c}
m \\
n \\
1 \\
\end{array} & \quad \begin{array}{c}
m \\
n \\
1 \\
\end{array} & \quad \begin{array}{c}
m \\
1 \\
\end{array}
\end{align*} \]

• What about a batch of 10 inputs?

\[ \begin{align*}
W \text{ (dims: m by n)} & \quad X \text{ (dims: n by 10)} & \quad \text{output (dims: m by 10)} \\
\begin{array}{c}
m \\
n \\
10 \\
\end{array} & \quad \begin{array}{c}
m \\
n \\
10 \\
\end{array} & \quad \begin{array}{c}
m \\
10 \\
\end{array}
\end{align*} \]
Benefit of matrices in batching

• GPU can process a whole batch in parallel!
  • In practice, we use the biggest batch size that will fit on our GPU (from last lecture)
• Example: Training duration of a CNN with GPU for different batch sizes
Adding a term (e.g. bias)

- \(W \times x:\)

\[
\begin{align*}
\text{W (dims: m by n)} & \quad \text{X (dims: n by 10)} & \quad \text{output (dims: m by 10)} \\
\begin{array}{c}
\text{m} \\
\text{n}
\end{array} & \quad \ast & \quad \begin{array}{c}
\text{10} \\
\text{n}
\end{array} & \quad = & \quad \begin{array}{c}
\text{m} \\
\text{10}
\end{array}
\end{align*}
\]

- \(+ b:\)

\[
\begin{align*}
\text{output (dims: m by 10)} & \quad \text{b (dims: m by 1)} \\
\begin{array}{c}
\text{m} \\
\text{10}
\end{array} & \quad + & \quad \begin{array}{c}
\text{m} \\
\text{1}
\end{array} & \quad = & \quad ???
\end{align*}
\]

- Can’t add matrices of different dimensions!
- What should we do?
Broadcasting

• Actually not a problem because of broadcasting!
• Broadcasting: implicitly replicating a tensor along some dimension to make math operations possible.
• NumPy, Tensorflow, PyTorch will all broadcast for you.

• Example: \((m, 10) + (1, 10) \not\equiv (m, 10) + m \times (1, 10)\)

  • \(+ b:\) output (dims: \(m\) by 10) \hspace{1cm} b (dims: \(m\) by 1)
Broadcasting

• Actually not a problem because of broadcasting!
• Broadcasting: implicitly replicating a tensor along some dimension to make math operations possible.
• NumPy, Tensorflow, PyTorch will all broadcast for you.

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
10 & 11 & 12 \\
\end{bmatrix} + [100 \quad 200 \quad 300] = \text{?}
\]
Broadcasting

• Actually not a problem because of broadcasting!
• Broadcasting: implicitly replicating a tensor along some dimension to make math operations possible.
• NumPy, Tensorflow, PyTorch will all broadcast for you.

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
10 & 11 & 12
\end{bmatrix}
+ [100 \ 200 \ 300] = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
10 & 11 & 12
\end{bmatrix}
+ \begin{bmatrix}
100 & 200 & 300 \\
100 & 200 & 300 \\
100 & 200 & 300 \\
100 & 200 & 300
\end{bmatrix} =
\]
Broadcasting

• Actually not a problem because of broadcasting!
• Broadcasting: implicitly replicating a tensor along some dimension to make math operations possible.
• NumPy, Tensorflow, PyTorch will all broadcast for you.

\[
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
10 & 11 & 12 \\
\end{bmatrix}
+ [100 
200 
300]
= 
\begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
10 & 11 & 12 \\
\end{bmatrix}
+ 
\begin{bmatrix}
100 & 200 & 300 \\
100 & 200 & 300 \\
100 & 200 & 300 \\
100 & 200 & 300 \\
\end{bmatrix}
= 
\begin{bmatrix}
101 & 202 & 303 \\
104 & 205 & 306 \\
107 & 208 & 309 \\
110 & 211 & 312 \\
\end{bmatrix}
\]
Broadcasting in NumPy

General Broadcasting Rules:

• When operating on two arrays, NumPy compares their shapes element-wise starting with the trailing dimensions.
• Two dimensions are compatible when
  • they are equal, or
  • one of them is 1
• Dimensions with size 1 are stretched or “copied” to match the other. The size of the resulting array is the maximum size along each dimension of the input arrays.
• **Arrays do not need to have the same number of dimensions, as long as the trailing dimensions are compatible.**

Link to NumPy documentation:
https://docs.scipy.org/doc/numpy/user/basics.broadcasting.html
Broadcasting in NumPy

• Example:
  • (m, n) array + (n,) array works
  • (m, n) array + (m,) array doesn’t work
  • (m, n) array + (m, 1) array works

• Which of the following examples work?
  A: (5, 3, 2) + (3, 2)
  B: (5, 3, 2) + (5, 2)
  C: (5, 3, 2) + (5, 3)
  D: (5, 3, 2) + (5, 1, 2)
  E: (5, 3, 2) + (1, 3, 2)
  F: (5, 3, 2) + (5, 3, 1)
  G: (5, 3, 2) + ()

Tensor: multi-dimensional array

Go to www.menti.com and use the code 2443 8157
Broadcasting in NumPy

• Example:
  • (m, n) array + (n,) array works
  • (m, n) array + (m,) array doesn’t work
  • (m, n) array + (m, 1) array works

• Which of the following examples work?
  A: (5, 3, 2) + (3, 2) = success!
  B: (5, 3, 2) + (5, 2) = failure 😞
  C: (5, 3, 2) + (5, 3) = failure 😞
  D: (5, 3, 2) + (5, 1, 2) = success!
  E: (5, 3, 2) + (1, 3, 2) = success!
  F: (5, 3, 2) + (5, 3, 1) = success!
  G: (5, 3, 2) + () = success!

Any questions?
Deep Learning Frameworks
History of deep learning frameworks

- **Lua**
  - Launched 2002 by academic researchers (who later went on to work for Facebook and Twitter)
  - Unified different ML algorithms into single framework
  - Use of niche Lua language limited adoption to dedicated researchers
  - No longer under active development

- **Python**
  - Launched 2007 by researcher at MILA (Montreal Institute for Learning Algorithms)
  - Essentially a GPU + symbolic differentiation backend for numpy
  - Cryptic errors, poor performance for larger models
  - No longer under active development

- **C++ (w/ models defined via text config files)**
  - Launched 2013 by a PhD student at Berkeley
  - Designed for vision models, very optimized.
  - Difficult to declare models that are more complicated than a linear chain of layers
  - Making custom layers requires writing C++ code...
  - No longer under active development

Did my PhD project in 2016 using this!!
History of deep learning frameworks

- **Lua**
  - Launched 2002 by academic researchers (who later went on to work for Facebook and Twitter)
  - Unified different ML algorithms into single framework
  - Use of niche Lua language limited adoption to dedicated researchers
  - No longer under active development

- **Python**
  - Launched 2007 by researcher at MILA (Montreal Institute for Learning Algorithms)
  - Essentially a GPU + symbolic differentiation backend for numpy
  - Cryptic errors, poor performance for larger models
  - No longer under active development

- **C++** (w/ models defined via text config files)
  - Launched 2013 by a PhD student at Berkeley
  - Designed for vision models, very optimized
  - Difficult to declare models that are more complicated than a linear chain of layers
  - Making custom layers requires writing C++ code...

Notice a common theme? What happened?
Current strong industrial players behind DL frameworks

Google \rightarrow TensorFlow

facebook \rightarrow PyTorch
We’ll be using TensorFlow
This choice isn’t hugely important

• Tensorflow and PyTorch have become increasingly similar in their designs, over the years
• They have about the same level of popularity
TensorFlow Demo

Collab Notebook
Recap

Neural networks as matrix operations

Batching and Broadcasting

Intro to TensorFlow

TensorFlow
Extra: GPU-Parallel Acceleration

• Multiple *streaming* multiprocessors (SMs)

[Link to research paper](https://www.researchgate.net/publication/23666656_Accelerating_Fibre_Orientation_Estimation_from_Diffusion_Weighted_Magnetic_Resonance_Imaging_Using_GPUs)
Extra: GPU-Parallel Acceleration

- Multiple **streaming multiprocessors (SMs)**
- Each SM has multiple **cores / streaming processors (SPs)**

https://www.researchgate.net/publication/236666656_Accelerating_Fibre_Orientation_Estimation_from_Diffusion_Weighted_Magnetic_Resonance_Imaging_Using_GPUs
**Single Instruction, Multiple Threads**

Programmer writes code for a single thread

All threads execute the same code, but can take different paths

Threads are grouped into a block

Threads within the same block can synchronize execution

Blocks are grouped into a grid

Blocks are independently scheduled on the GPU, can execute in any order

A kernel is executed as a grid of blocks and threads.

A warp is the basic schedule unit in kernel execution.

A warp consists of 32 threads.

A thread block consists of multiple warps.

Each cycle, a warp scheduler selects one ready warps and dispatches the warps to CUDA cores to execute.