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

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

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

which means:
• $m$ sets of weight vectors (or a weight matrix)
• $m$ biases
**Fully connected layer with multiple outputs**

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

- \( W = \) matrix of weights (3\times7 in this example)
- \( x = \) input as a vector (7\times1 in this example)
- \( b = \) vector bias (3\times1)
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: $\Delta \mathbf{w}_{i,j} = -\alpha \cdot \frac{\partial L}{\partial w_{i,j}}$

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

10x784 matrix of weights

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

\[
\begin{bmatrix}
\frac{\partial L}{\partial w_{1,1}} & \frac{\partial L}{\partial w_{1,2}} & \cdots & \frac{\partial L}{\partial w_{1,784}} \\
\vdots & \vdots & \ddots & \vdots \\
\frac{\partial L}{\partial w_{10,1}} & \frac{\partial L}{\partial w_{10,2}} & \cdots & \frac{\partial L}{\partial w_{10,784}}
\end{bmatrix}
\]

Jacobian matrix: matrix of all first-order partial derivatives for a vector-valued function.
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 $a, b \in \mathbb{R}^{1,000,000}$

• 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

Arithmetic logic unit

CPU v/s GPU

CPU

output

input

ALU

ALU

ALU

ALU

Decode

Fetch

output

Vector operations (SSE / AVX)

GPU: specialized accelerator

ALU

ALU

ALU

ALU

ALU

ALU

ALU

ALU

ALU

ALU

ALU

ALU

ALU

ALU

Write back

Fetch

Decode

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)

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

Device
2 x AMD Opteron 6168
i7-7500U
GeForce 940MX
GeForce 1070

<table>
<thead>
<tr>
<th>Device</th>
<th>Speed of training, examples/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 x AMD Opteron 6168</td>
<td>440</td>
</tr>
<tr>
<td>i7-7500U</td>
<td>415</td>
</tr>
<tr>
<td>GeForce 940MX</td>
<td>1190</td>
</tr>
<tr>
<td>GeForce 1070</td>
<td>6500</td>
</tr>
</tbody>
</table>

From: 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

\[ W (\text{dims: } m \times n) \times X (\text{dims: } n \times 1) = \text{output (dims: } m \times 1) \]

• What about a batch of 10 inputs?

\[ W (\text{dims: } m \times n) \times X (\text{dims: } n \times 10) = \text{output (dims: } m \times 10) \]
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)

- **$W \times x$:**
  
  ![Diagram](image)

  - 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: \((m, 10) + (1, 10) \rightarrow (m, 10) + m \times (1, 10)\)

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

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

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

Broadcasting
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](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

Join at menti.com | use code 7935 1000
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

**Torch**
- 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

**Theano**
- 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

**Caffe**
- 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!!*
The 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 led to 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

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

Google → TensorFlow

Facebook → 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
Extra: GPU-Parallel Acceleration

- Multiple *streaming multiprocessors* (SMs)

[Diagram of Architecture of a CUDA-capable GPU]

https://www.researchgate.net/publication/236666656_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
Extra: Programming model - SIMT

**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

Extra: Programming model - SIMT