Autodiff Bootcamp: new_grad

Kinori Rosnow, Anurag Katakka, Shriti Priya, David Park
Backpropagate Loss

1. Forward
2. Calculate Loss
3. Pass Gradient with respect to output
4. Update Parameters
5. Continue

Output = f3( f2( f1( x ) ) )
Single Layer Backward: Linear

\[
\frac{dL}{dx} = \frac{dL}{dy} \frac{dy}{dx}
\]

\[
\frac{dL}{dW} = \frac{dL}{dy} \frac{dy}{dW}
\]

\[
\frac{dL}{db} = \frac{dL}{dy} \frac{dy}{db}
\]

Update W

Update b
How does Pytorch take derivatives and backpropagate?

Auto-differentiation:

- All of the functions can be rewritten into basic operations
  - True for all computer based calculations
- Sequence of operations instead of a layers
- Each operation is differentiable

\[ z = Wx + b \]
\[ y = Wx \]
\[ z = y + b \]
Operational Order

Derive matrix multiplication

Derive addition
Deep Learning Computation Actually

- Operations are monotonically ordered
- 2 methods for backprop
  - Traverse directed acyclic graph (DAG)
  - Take advantage of ordering - clever gradient storage
- Pytorch's Autograd - tensor class
  - Computational DAG
  - Backpropagation = graph traversal
- new_grad - memory buffer class
  - Computational list
  - Backpropagation = iterate backwards
Operation List Implementation

Memory Buffer

\[ \frac{dL}{dx} \]
\[ \frac{dL}{dW} \]
\[ \frac{dL}{dy} \]
\[ \frac{dL}{db} \]
\[ \frac{dL}{dz} \]

Derive addition

\[ \frac{dL}{dy} = \frac{dL}{dz} \cdot \frac{dz}{dy} \]
\[ \frac{dL}{db} = \frac{dL}{dz} \cdot \frac{dz}{db} \]

Derive matrix multiplication

\[ \frac{dL}{dx} = \frac{dL}{dy} \cdot \frac{dy}{dx} \]
\[ \frac{dL}{dW} = \frac{dL}{dy} \cdot \frac{dy}{dW} \]