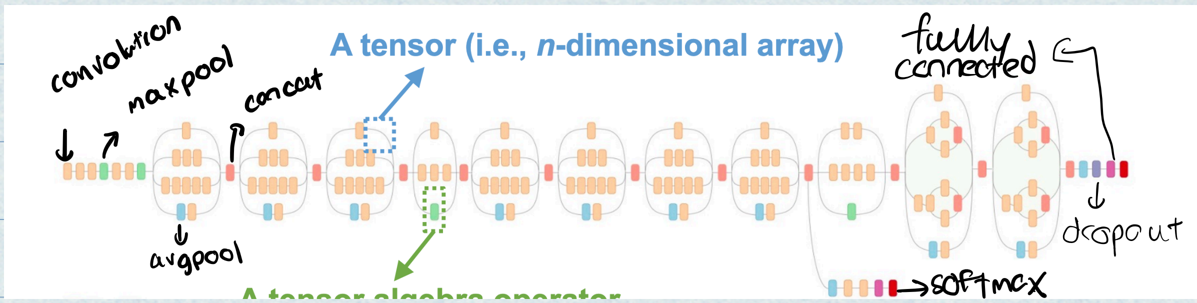
Agenda:

1) Review of Deep Learning, SGD, Backprop
2) Automatic Differentiation
3) Autograd in pytorch
4) CNNs / AlexNet + challenges
5) Basic intro to parallel training
6) Intro to Project 0

Part 1: Review of DNNs and Backprop

- What is a DNN? Collection of simple trainable mathematical units that work together to solve complex tasks



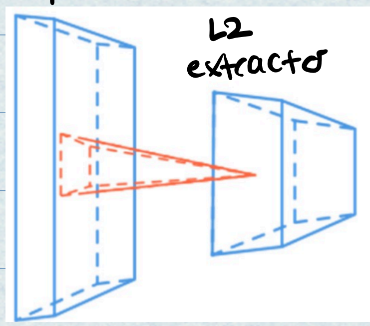
- DNNs are a series of TENSOR ALGEBRA OPERATORS (e.g. convolution or matrix mul) over tensors (n-d arrays)

- DNN Training overview

Input Data

→

layer1 extractor



Predictor

$$\longrightarrow \hat{y}_i = \frac{1}{1 + \exp(-w^T x_i)}$$

Objective: $L(w) = \sum_{i=1}^{n} l(y_i, \hat{y}_i) + \lambda \|w\|^2$

gradient update

Training update: $w \leftarrow w - \boxed{\eta \nabla_w L(w)}$

- General Training Loop for DNNs

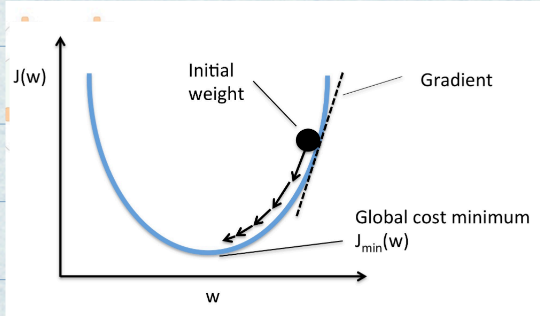
1) • Forward Propagation: apply model to batch of inputs, run calculations through ops

2) • Backward prop: run model in reverse to produce error for each trainable weight

3) • Weight update: use loss to update model weight for next iter.

- Gradient Descent:
  - For each learnable parameter, we calculate:

$$\frac{\partial L(w)}{\partial w_i}$$



updates gradually lead weight to value minimizing cost

- update step:

$$w_i = w_i - \eta \frac{\partial L(w)}{\partial w_i} = w_i - \frac{\eta}{N} \sum_{j=1}^{N} \boxed{\frac{\partial l_j(w)}{\partial w_i}}$$

gradients of individual samples

all training samples

- Stochastic Gradient Descent (SGD):
  - Too expensive to compute gradients for each training sample
  - Imagenet-22K has 14 mil. images

* sgd: divide dataset into BATCHES

$$w_i - \frac{\eta}{N} \sum_{j=1}^{N} \frac{\partial l_i(w)}{\partial w_i} \approx w_i - \frac{\eta}{b} \sum_{j=1}^{b} \frac{\partial l_i(w)}{\partial w_i}$$

batchsize

- instead of making each update correspond to an iteration of ENTIRE DATASET - update per batch

- Reminder of Backprop:
  - sum rule:

$$\frac{\partial (f(x) + g(x))}{\partial x} = \frac{\partial f(x)}{\partial x} + \frac{\partial g(x)}{\partial x}$$

  - product rule:

$$\frac{\partial (f(x) g(x))}{\partial x} = \frac{\partial f(x)}{x} \cdot g(x) + \frac{\partial g(x)}{\partial x} \cdot f(x)$$

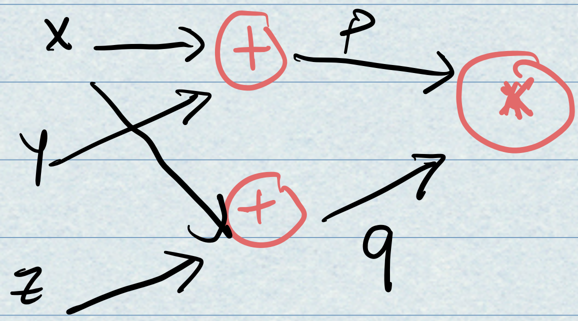
- chain rule:

$$\frac{\partial f(g(x))}{\partial x} = \frac{\partial f(y)}{y} \cdot \frac{\partial g(x)}{\partial (x)}$$

- Example:

D a

$f(x, y, z) = (x+y) \cdot (x+z)$



→ each node is an intermediate variable

→ DAG will have some kind of topological sort

- simple example of backprop

$x = -2 \quad y = 5 \quad z = -4$ (compute $\partial f / \partial x$)

$p = x + y = 3 \qquad \partial p / \partial x = 1$

$q = x + z = -6 \qquad \partial q / \partial x = 1$

$$f = p \cdot q \rightarrow \frac{\partial f}{\partial x} = \frac{\partial p}{\partial x} \cdot q + \frac{\partial q}{\partial x} \cdot p$$

$$= 1 \cdot (-6) + 1 \cdot (3) = -3$$

compute $\partial f / \partial y$:

$p = x + y \quad : \quad \partial p / \partial y = 1$

$q = x + z \quad : \quad \partial q / \partial y = 0$

$$f = (p \cdot q) \rightarrow \frac{\partial f}{\partial y} = \frac{\partial p}{\partial y} \cdot q + \frac{\partial q}{\partial y} \cdot p$$

$$= 1(-6) + 0(3) = -6$$

compute $\partial f / \partial z$:

$p = x + y$: $\partial p / \partial z = 0$

$q = x + z$: $\partial q / \partial z = 1$

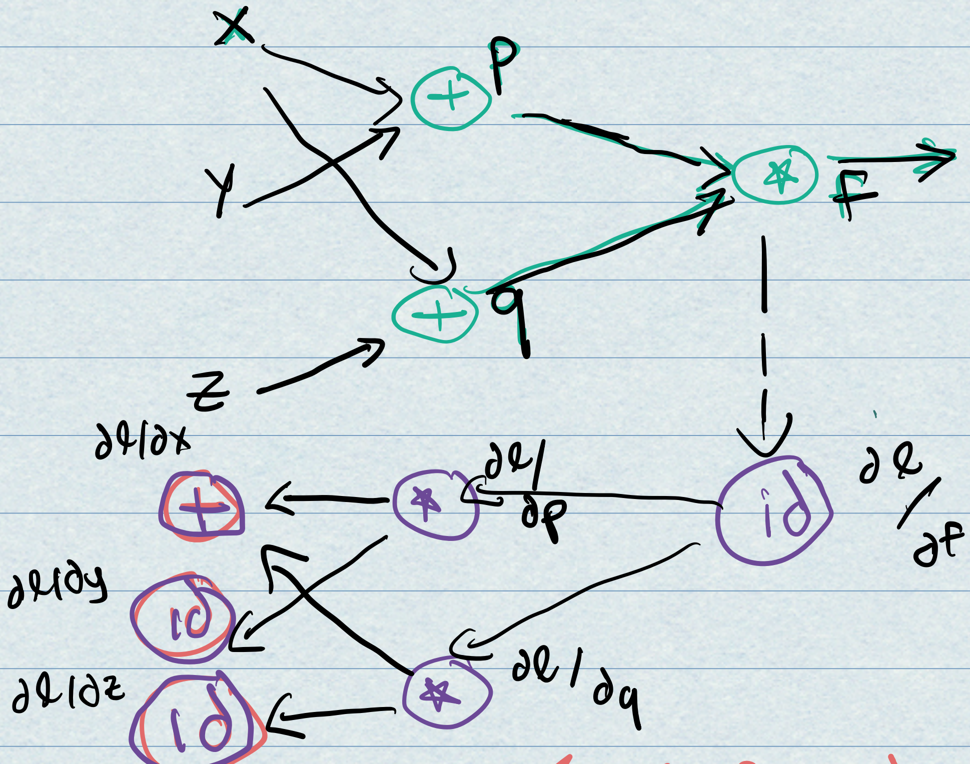
$$f = p \cdot q \rightarrow \frac{\partial f}{\partial z} = \frac{\partial p}{\partial z} \cdot q + \frac{\partial q}{\partial z} \cdot p$$

$$= 0(-6) + 1(3) = 3$$

- Problems:
  - If there are n input variables - we need n forward passes to compute gradient w.r.t each input
  - DL models have BILLIONS - TRILLIONS

- solution: AUTODIFF
  - idea: for each node v, introduce adjoint node $\bar{v}$ corresponding to gradient of output to this node: $\frac{\partial f}{\partial v}$
  - compute gradients in reverse topo order to save computation

→ lets recompute $\partial f/\partial x$, $\partial f/\partial y$, $\partial f/\partial z$



→ what is $\partial l/\partial f$? 1 for id loss

→ $\partial l/\partial q$?

$$\frac{\partial l}{\partial q} = \frac{\partial l}{\partial f}\cdot\frac{\partial f}{\partial q} = 1\cdot p = 3$$

→ $\partial l/\partial p$?

$$\frac{\partial l}{\partial p} = \frac{\partial l}{\partial f}\cdot\frac{\partial f}{\partial p} = 1\cdot q = -6$$

→ $\partial l / \partial x$

$$\frac{\partial l}{\partial x} = \frac{\partial l}{\partial p} \cdot \frac{\partial p}{\partial x} + \frac{\partial l}{\partial q} \cdot \frac{\partial q}{\partial x}$$

$$= \frac{\partial l}{\partial p} + \frac{\partial l}{\partial q} = 3 - 6 = -3$$

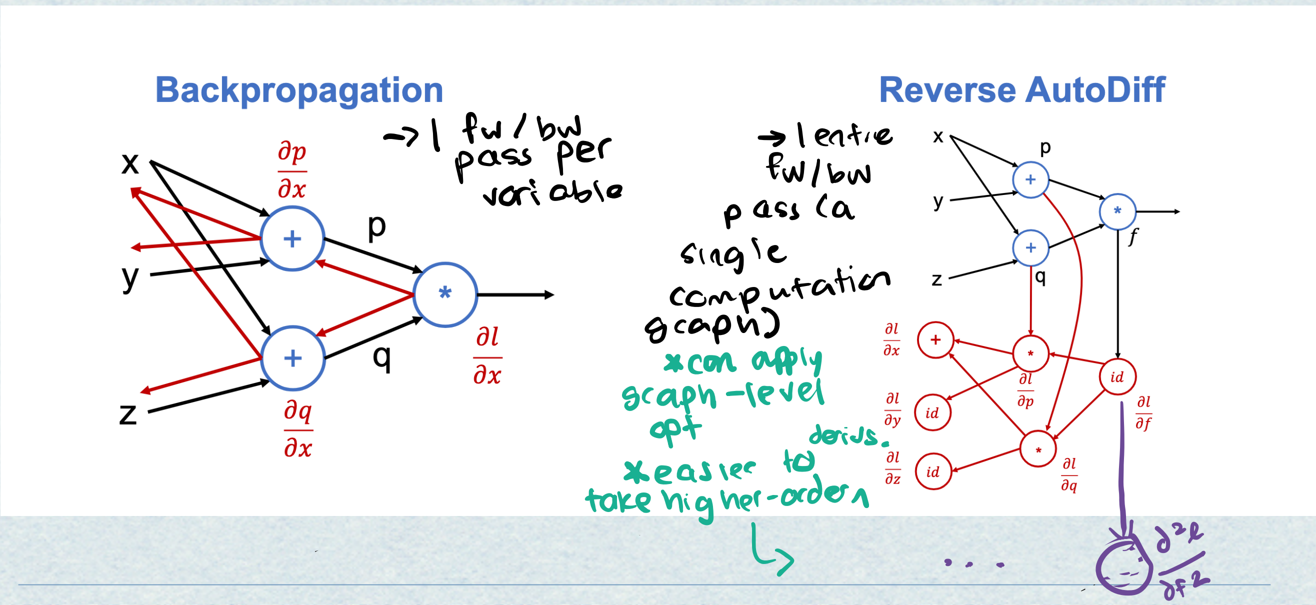
→ $\partial l / \partial y$

$$\frac{\partial l}{\partial y} = \frac{\partial l}{\partial p} \cdot \frac{\partial p}{\partial y} = -6 \cdot 1 = -6$$

→ $\partial l / \partial z$

$$\frac{\partial l}{\partial z} = \frac{\partial l}{\partial q} \cdot \frac{\partial q}{\partial z} = 3 \cdot 1 = 3$$

- MAIN Differences btwn backprop and Autodiff:

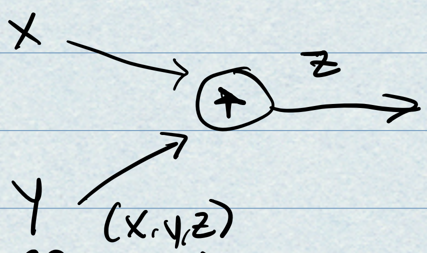


- Pseudocode for autograd?

```
class ComputationalGraph(object):
    #...
    def forward(inputs):
        # 1. [pass inputs to input gates...]
        # 2. forward the computational graph:
        for gate in self.graph.nodes_topologically_sorted():
            gate.forward()
        return loss # the final gate in the graph outputs the loss
    def backward():
        for gate in reversed(self.graph.nodes_topologically_sorted()):
            gate.backward() # little piece of backprop (chain rule applied)
        return inputs_gradients
```

*Topolgical sort allows you to avoid recomputation

- Example multiply gate:



(x,y,z) are scalars

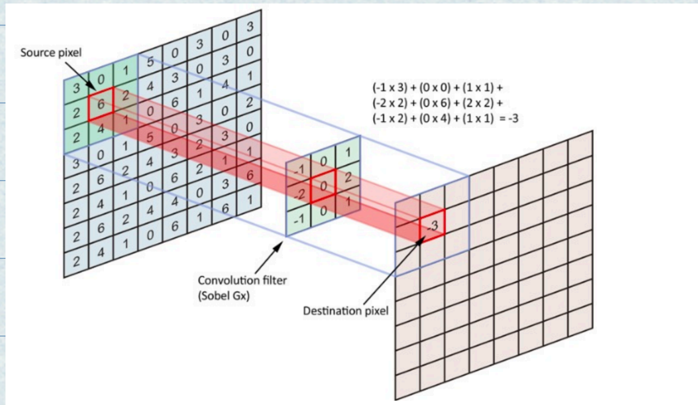
```
class MultiplyGate(object):
    def forward(x,y):
        z = x*y
        self.x = x # must keep these around!
        self.y = y
        return z
    def backward(dz):
        dx = self.y * dz # [dz/dx * dL/dz]
        dy = self.x * dz # [dz/dy * dL/dz]
        return [dx, dy]
```

- short demo of Autograd in Python.

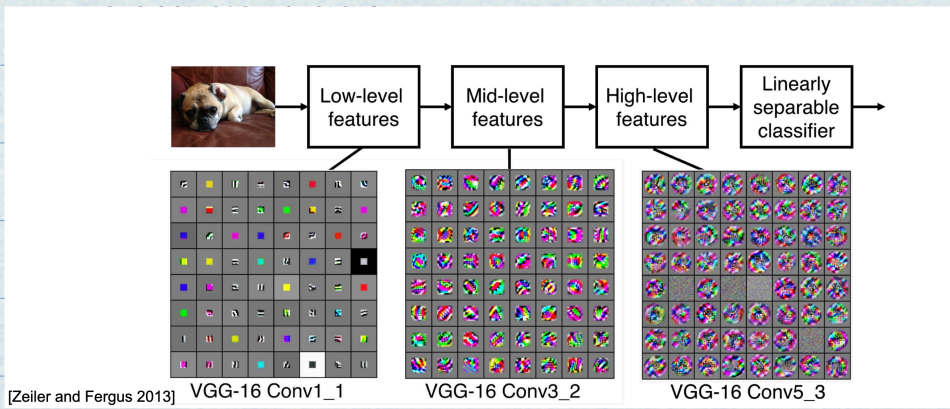
CNNs / AlexNet

- classification, segmentation
  self-driving, synthesis

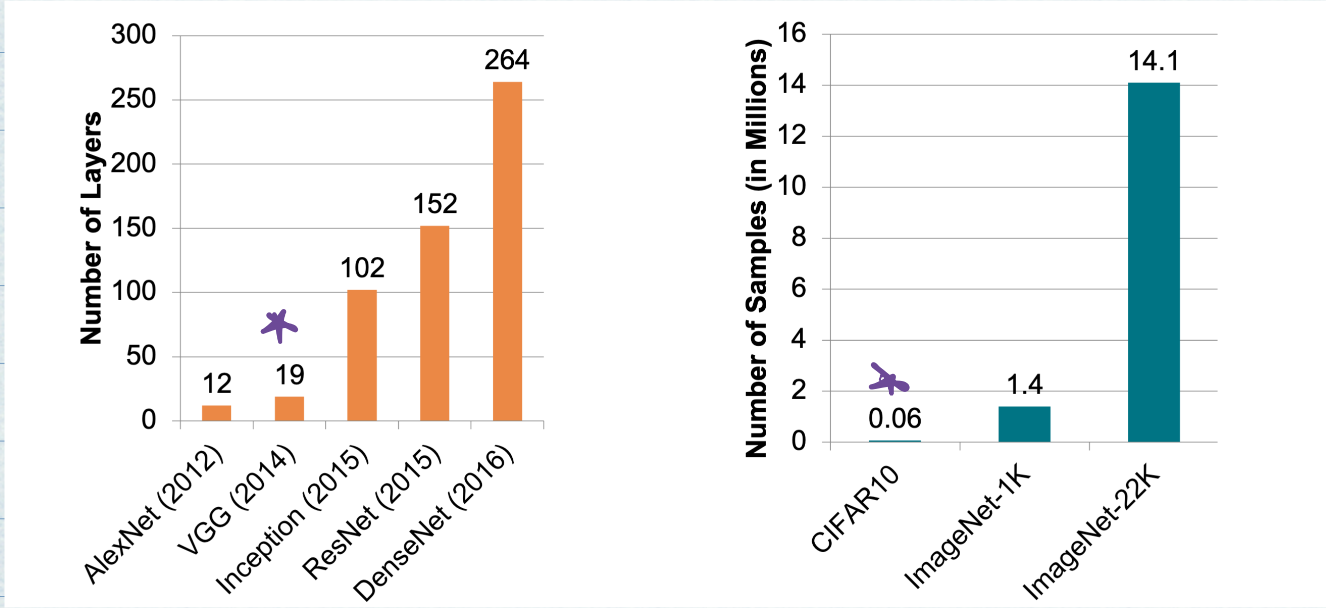
- Recap of Convolution
  - convolve filter w/ image: slide over image spatially and compute dot product



- CNNs = sequence of convolutional layers - interspersed by pooling, normalization and activations

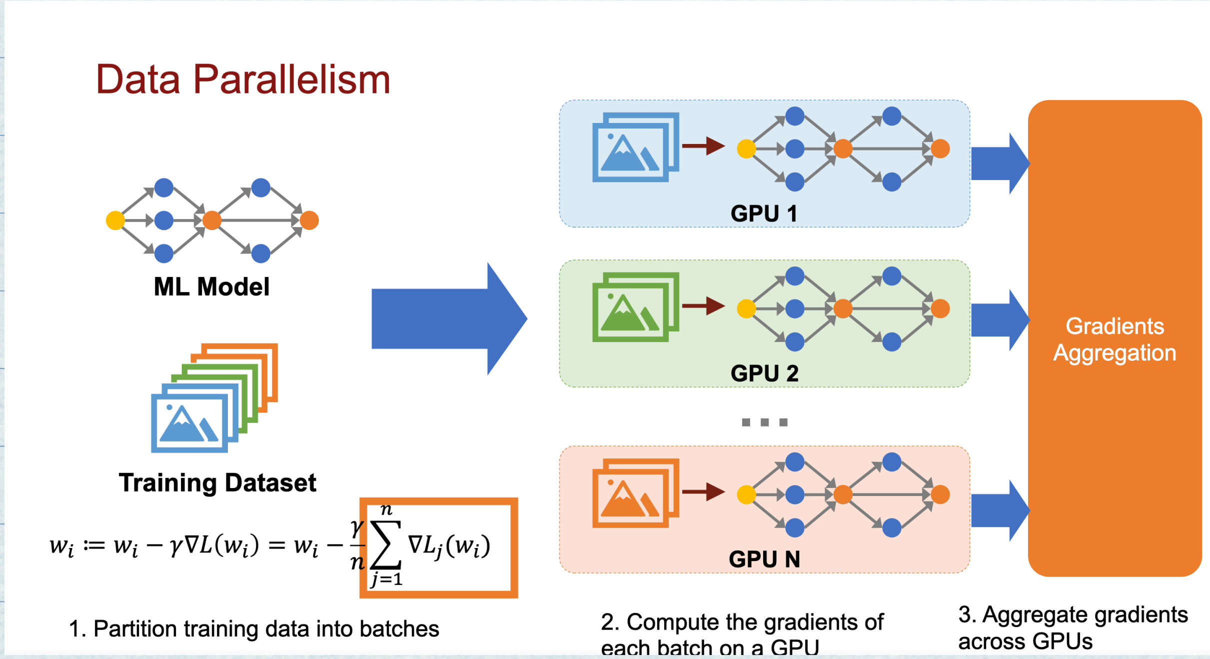


- MLsys challenges for CNN:
  - higher and higher computational costs - convolutions are extremely compute-intensive
  - memory: high-res images cannot fit on a single GPU



- AlexNet:
  - 90 epochs of 1.2 million training images
  - 5-6 days on NVIDIA GTX 580 3GB GPUs

- Short Intro to HW: training vgg 16 on cifar10



Credits for this lecture (figures and content):
- “Intro to Deep Learning Lecture” from CMU’s 15-849: https://www.cs.cmu.edu/~zhihaoj2/15-849/slides/02-deep-neural-networks.pdf
- Parallelism image from “Intro to Deep Learning Systems” from CMU 15-849: https://www.cs.cmu.edu/~zhihaoj2/15-849/slides/03-deep-learning-systems.pdf