Name:

CMSC 838B & 498Z: Differentiable Programming

Tues/Thur 12:30pm – 1:45pm
IRB 4105 (T) & IRB 5105 (R)
http://www.cs.umd.edu/class/fall2021/cmsc838b

Ming C. Lin

IRB 5162

lin@cs.umd.edu

http://www.cs.umd.edu/~lin

Office Hours: After Class or By Appointment

M. C. Lin

Optimization

To solve **optimization** problems using gradient methods we need to compute the gradients (derivatives) of the objective w.r.t. the parameters

●In neural nets we're talking about the gradients of the loss function w.r.t. the parameters θ: $\nabla L = \frac{\partial L}{\partial \theta}$

●3 ways to compute derivatives: Symbolic, Finite Difference, and ***Automatic Differentiation***

M. C. Lin

Name:

Automatic Differentiation (AD)

- A method to get exact derivatives efficiently, by storing information as you go forward that you can reuse as you go backwards
  - Takes code that computes a function and uses that to compute the derivative of that function
  - The goal isn't to obtain closed-form solutions, but to be able to write a program that efficiently computes the derivatives.

M. C. Lin

Differentiation and Programming

Example (Math)

$$x = ?$$
$$y = ?$$
$$a = xy$$
$$b = \sin(x)$$
$$z = a + b$$

Example (code)

```
x = ?
y = ?
a = x * y
b = sin (x)
z = a +  b
```

$$\frac{\partial x}{\partial t} = ?$$
$$\frac{\partial y}{\partial t} = ?$$
$$\frac{\partial a}{\partial t} = x\frac{\partial y}{\partial t} + y\frac{\partial x}{\partial t}$$
$$\frac{\partial b}{\partial t} = \cos(x)\frac{\partial x}{\partial t}$$
$$\frac{\partial z}{\partial t} = \frac{\partial a}{\partial t} + \frac{\partial b}{\partial t}$$

Name:

Forward Mode AD

- To translate using the rules we simply replace each primitive operation in the original program by its differential analogue
- The order of computation remains unchanged: if a statement K is evaluated before another statement L, then the differential analogue of K is evaluated before the analogue statement of L
- This is ***Forward-mode Automatic Differentiation***

M. C. Lin

Backward AD: Reversing the Chain Rule

$$\frac{\partial s}{\partial u} = \sum_{i}^{N} \frac{\partial w_i}{\partial u}\frac{\partial s}{\partial w_i}$$

$$x = ?$$
$$y = ?$$
$$a = x\,y$$
$$b = \sin(x)$$
$$z = a + b$$

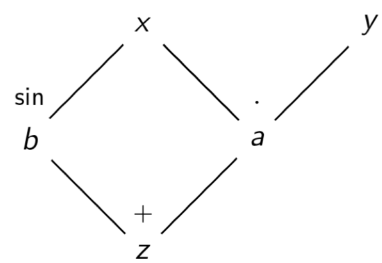
$$\frac{\partial s}{\partial z} = ?$$
$$\frac{\partial s}{\partial b} = \frac{\partial z}{\partial b}\frac{\partial s}{\partial z} = \frac{\partial s}{\partial z}$$
$$\frac{\partial s}{\partial a} = \frac{\partial z}{\partial a}\frac{\partial s}{\partial z} = \frac{\partial s}{\partial z}$$
$$\frac{\partial s}{\partial y} = \frac{\partial a}{\partial y}\frac{\partial s}{\partial a} = x\,\frac{\partial s}{\partial a}$$
$$\frac{\partial s}{\partial x} = \frac{\partial a}{\partial x}\frac{\partial s}{\partial a} + \frac{\partial b}{\partial x}\frac{\partial s}{\partial b} = y\,\frac{\partial s}{\partial a} + \cos(x)\,\frac{\partial s}{\partial b} = (y + \cos(x))\,\frac{\partial s}{\partial z}$$

M. C. Lin

Name:

Visualizing Dependencies

- Differentiating in reverse can be quite mind-bending: instead of asking what input variables an output depends on, we have to ask what output variables a given input variable can affect.
- We can see this visually by drawing a dependency graph of the expression:



M. C. Lin

Gradient Descent

- Define total loss as $\mathcal{L} = \sum_{(\boldsymbol{x},y)\in \boldsymbol{D}} \ell(g(\boldsymbol{x}, \boldsymbol{\theta}), y)$ for some loss function ℓ, dataset D, and model g, with learnable parameters Θ
- Define how many passes over the data to make (each one known as an Epoch)
- Define a learning rate η

Gradient Descent updates the parameters Θ by moving them in the direction of the negative gradient with respect to the **total loss** $\mathcal{L}$ by the learning rate η multiplied by the gradient:

for each Epoch:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}$$

M. C. Lin

Name:

Gradient Descent

- Gradient Descent has good statistical properties (very low variance)
- But is very data inefficient (particularly when data has many similarities)
- Doesn't scale to effectively infinite data (e.g. with data augmentation)

M. C. Lin

Stochastic Gradient Descent (SGD)

- Define loss function ℓ, dataset D, and model g, with learnable parameters Θ
- Define how many passes over the data to make (each one known as an Epoch)
- Define a learning rate η

Stochastic Gradient Descent (SGD) updates the parameters Θ by moving them in the direction of the negative gradient with respect to the loss of a single item ℓ by the learning rate η, multiplied by the gradient:

for each Epoch:

for each $(\boldsymbol{x}, y) \in \boldsymbol{D}$:

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \ell$$

M. C. Lin

Name:

Stochastic Gradient Descent (SGD)

- Stochastic Gradient Descent has poor statistical properties (very high variance)
- Why works?
  - We don't need to check all the training examples to get an idea about the direction of decreasing slope. By analyzing only 1 example at a time and following its slope (gradient), we can reach a point very close to the actual minimum
- Not computationally efficient enough (poor utilization of resources w.r.t. vectorization)

M. C. Lin

Mini-Batch Stochastic Gradient Descent

- Define a batch size b
- Define batch loss $\mathcal{L}_b = \sum_{(\boldsymbol{x},y)\in \boldsymbol{D}_b} \ell(g(\boldsymbol{x},\boldsymbol{\theta}), y)$ as for some loss function ℓ & model g with learnable parameters Θ. D_b is a subset of dataset D of cardinality b
- Define how many passes (Epochs) over the data to make
- Define a learning rate η

Mini-batch Stochastic Gradient Descent (SGD) updates parameters Θ by moving them in the direction of the negative gradient with respect to the loss of a mini-batch D_b, $\mathcal{L}_b$ by the learning rate η, multiplied by the gradient:

partition the dataset $\boldsymbol{D}$ into an array of subsets of size b

```
for each Epoch:
    for each D_b ∈ partitioned(D):
        θ ← θ − η∇_θ L_b
```

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \eta \nabla_{\boldsymbol{\theta}} \mathcal{L}_b$$

M. C. Lin

Name:

Mini-Batch Stochastic Gradient Descent

- Mini-batch Stochastic Gradient Descent has reasonable statistical properties (much lower variance than SGD)
- Allows for computationally efficiency (good utilization of resources)
- Ultimately we would normally want to make our batches as big as possible for lower variance gradient estimates, but:
  - Must still fit in RAM (e.g. on the GPU)
  - Must be able to maintain throughput (e.g. pre-processing on the CPU; data transfer time)

M. C. Lin

Learning Rates

- Choice of learning rate is extremely important
- But we have to reason about the 'loss landscape'
  - Most convergence analysis of optimization algorithms assumes a convex loss landscape
    - Easy to reason about
    - Can be shown that (S)GD will converge to the optimal solution for a variety of learning rates
    - Can give insights into potential problems in the non-convex case
  - Deep Learning is highly non-convex
    - Many local minima
    - Plateaus
    - Saddle points
    - Symmetries (permutation, etc)
    - Certainly no single global minima

M. C. Lin

Name:

Accelerated Gradient Methods

- Accelerated gradient methods use a *leaky* average of the gradient, rather than the instantaneous gradient estimate at each time step

- A physical analogy would be one of the momentum a ball picks up rolling down a hill...

- This helps address the *GD failure modes, but also helps avoid getting stuck in local minima

M. C. Lin

Momentum I

- It's common for the 'leaky' average (the 'velocity', v_t) to be a weighted average of the instantaneous gradient g_t and the past velocity[1]:

$$v_t = \beta v_{t-1} + g_t$$

where $\beta \in [0, 1]$ is the 'momentum'

[1]There are quite a few variants – here the PyTorch variant

M. C. Lin

Name:

Momentum II

- The momentum method allows to accumulate velocity in directions of low curvature that persist across multiple iterations
- This leads to accelerated progress in low curvature directions compared to gradient descent

M. C. Lin

MB-SGD with Momentum

- Learning with momentum on iteration t (batch at t denoted by $b(t)$) is given by:

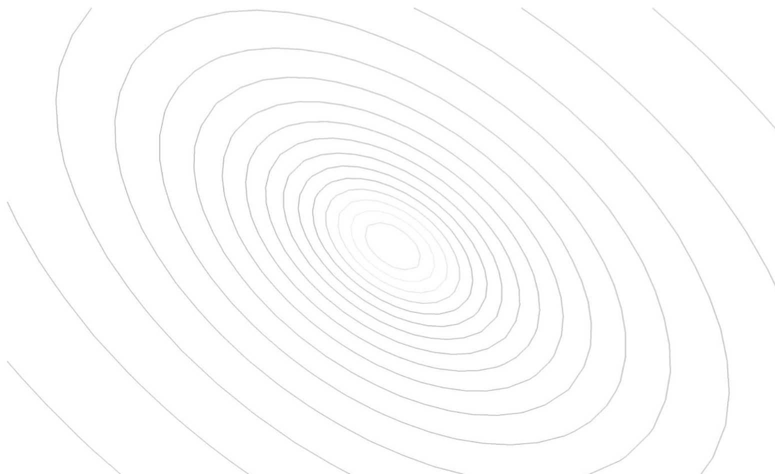
$$\mathbf{v}_t \leftarrow \beta \mathbf{v}_{t-1} + \nabla_{\boldsymbol{\theta}} \mathcal{L}_{b(t)}$$
$$\boldsymbol{\theta}_t \leftarrow \boldsymbol{\theta}_{t-1} - \eta \mathbf{v}_t$$

$\beta = 0.9$ *is a good choice for the momentum parameter*

M. C. Lin

Name:

SGD with Momentum - potentially better convex convergence



Learning Rates

- In practice you want to decay your learning rate over time
- Smaller steps will help get closer to the minima
- But don't do it too early, else might get stuck
- Something of an art form!

M. C. Lin

Name:

Reduce LR on Plateau

- Common Heuristic approach:
  - – if the loss hasn't improved (within some tolerance) for *k* epochs then drop the LR by a factor of 10
- Remarkably powerful!

M. C. Lin

Cyclic Learning Rates

- Worried about getting stuck in a non-optimal local minima?
- Cycle the learning rate up and down (possibly annealed), with a different LR on each batch
- See https://arxiv.org/abs/1506.01186

M. C. Lin

Name:

More Advanced Optimizers

- **Adagrad**
  - Decrease learning rate dynamically per weight.
  - Squared magnitude of the gradient (2nd moment) used to adjust how quickly progress is made - weights with large gradients are compensated with a smaller learning rate.
  - Particularly effective for sparse features.
- **RMSProp**
  - Modify Adagrad to decouple learning rate from gradient magnitude scaling
  - Incorporates leaky averaging of squared gradient magnitudes
  - LR would typically follow a predefined schedule
- **Adam**
  - Essentially takes all the best ideas from RMSProp and SDG+Momentum
  - Bias corrected momentum and second moment estimation
  - It might still diverge (or be non optimal, even in convex settings)...
  - LR is still a hyperparameter (you might still schedule)

M. C. Lin

Take-away Messages

- The loss landscape of a deep network is complex to understand (and is far from convex)
- If you're in a hurry to get results use Adam
- If you have time, then use SGD (with momentum) and work on tuning learning rates
- If you're implementing something from a paper, then follow what they did!

For more about Numerical Optimization: CMSC 764

M. C. Lin