Momentum Methods

Part II Roadmap

- Part I – Linear Algebra (units 1-12) $Ac = b$

linearize

line search

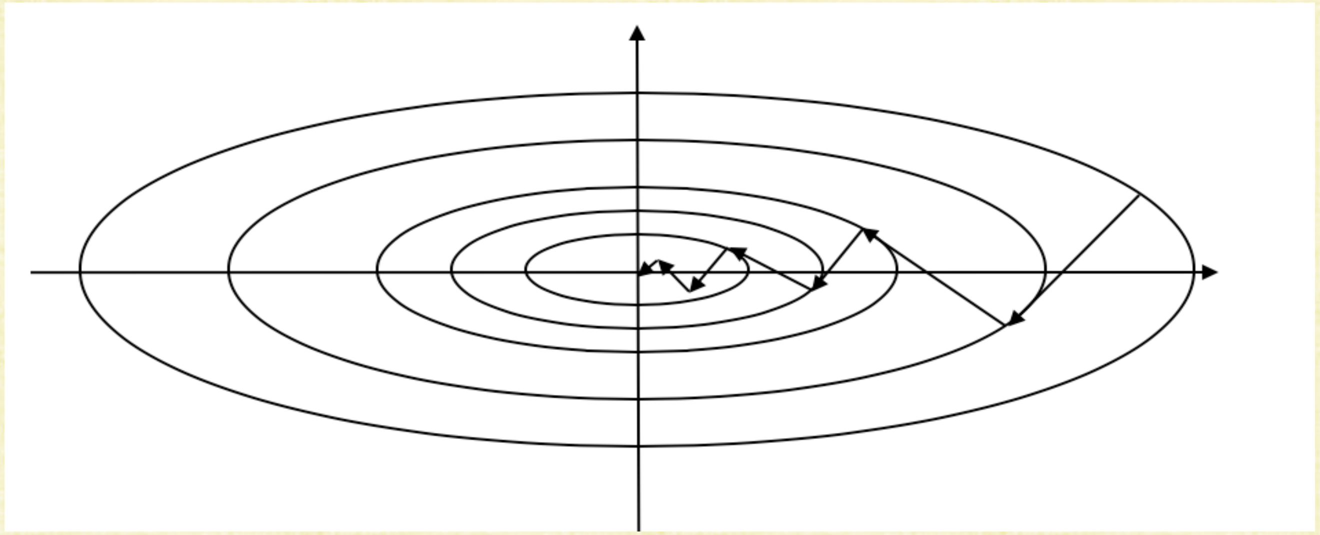
- Part II – Optimization (units 13-20)
  - (units 13-16) Optimization -> Nonlinear Equations -> 1D roots/minima
  - (units 17-18) Computing/Avoiding Derivatives
  - (unit 19) Hack 1.0: "I give up" $H = I$ and J is mostly 0 (descent methods)
  - (unit 20) Hack 2.0: "It's an ODE!?" (adaptive learning rate and momentum)

Theory

Methods

Path through Parameter Space

- Optimization solvers iteratively update the state variable c at each iteration
- For difficult problems (such as neural network training), this is typically done via a 1D line search at each iteration
- The union of all such line searches can be thought of as a path through parameter space



Continuous Path vs Discrete Path

- Each iteration is a discrete jump from one point to another, and connecting them with a 1D line segment is merely a visualization
- In the limit as the size of the segments goes to zero (and the number of iterations goes to infinity), one obtains a continuous path

- Can parameterize this path/curve with a scalar t (typically called <u>time</u>)
- Then $c(t)$ is a continuous path in parameter space ($c(t)$ is a <u>position</u>)
  - Changing the value of t moves the position $c(t)$ along the path
- Differentiating the continuous path gives a time varying <u>velocity</u>: $\frac{dc}{dt}(t)$ or $c'(t)$

Ordinary Differential Equations (ODEs)

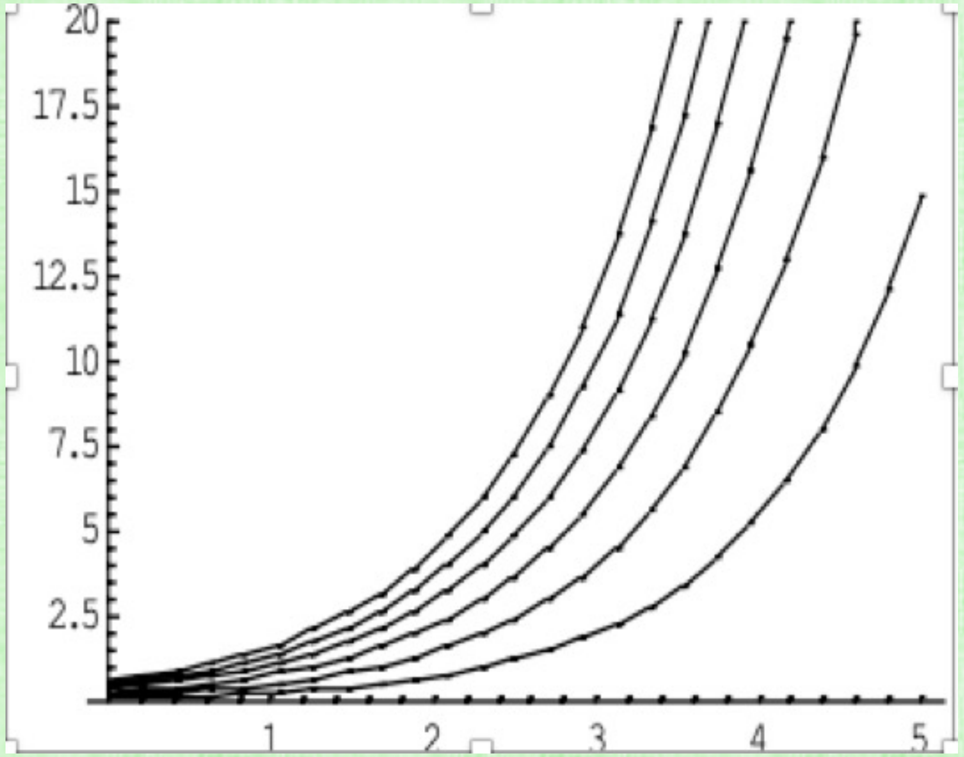
- ODEs are equations that describe rates of change
  - For example, $\frac{dc}{dt}(t) = f(t, c(t))$ states that the parameter space velocity is $f(t, c(t))$
- "Solving" an ODE means finding a function with rates of change described by the ODE
  - Given the velocity along the curve $\frac{dc}{dt}(t) = f(t, c(t))$, find the curve $c(t)$ itself
- Consider a (greedy) steepest decent path which always follows the steepest downhill direction for a cost function $\hat{f}(c)$
  - A suitable velocity is any (positive) scalar multiple of $-\nabla \hat{f}(c(t))$
  - This leads to an ODE: $\frac{dc}{dt}(t) = -\nabla \hat{f}(c(t))$

Gradient Flow

- The ODE for gradient flow is: $\frac{dc}{dt}(t) = -\nabla \hat{f}\big(c(t)\big)$
- Or (in more detail): $\begin{pmatrix} \frac{dc_1}{dt}(t) \\ \frac{dc_2}{dt}(t) \\ \vdots \\ \frac{dc_n}{dt}(t) \end{pmatrix} = \begin{pmatrix} -\frac{\partial \hat{f}}{\partial c_1}\big(c(t)\big) \\ -\frac{\partial \hat{f}}{\partial c_2}\big(c(t)\big) \\ \vdots \\ -\frac{\partial \hat{f}}{\partial c_n}\big(c(t)\big) \end{pmatrix}$
- $c(t)$ is a function of time t that evolves/changes based on the local gradient of the cost function, $-\nabla \hat{f}\big(c(t)\big)$
- This path follows the direction of steepest descent

Families of Solutions

- ODEs are <u>initial value problems</u>: the solution depends on the initial (starting) condition



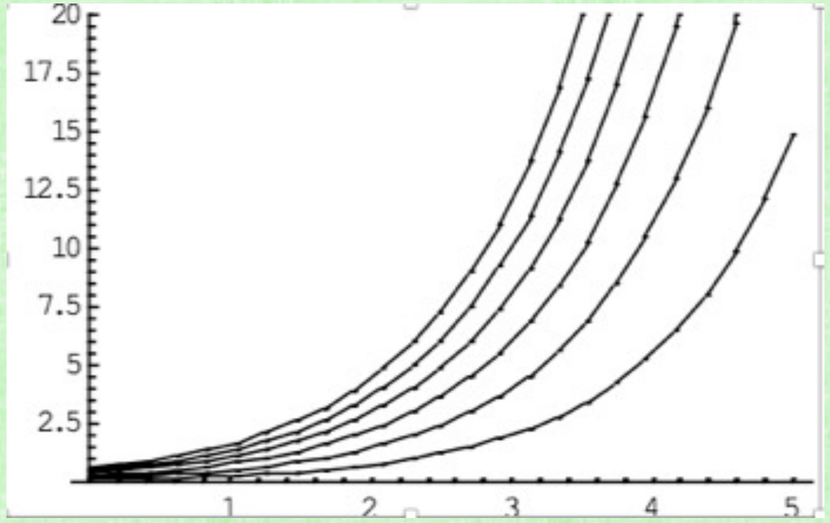
- E.g. $c' = c$ or $\frac{dc}{dt} = c$ or $\frac{dc}{c} = dt$
- $\int_{c_o}^{c} \frac{1}{c} dc = \int_{t_o}^{t} dt$ or $\ln c - \ln c_o = t - t_o$
- $\ln \frac{c}{c_o} = t - t_o$ or $\frac{c}{c_o} = e^{t-t_o}$ or $c = c_o e^{t-t_o}$
- Solution $c(t) = c_o e^{t-t_o}$ depends on the initial condition $c(t_o) = c_o$
- The figure shows solutions for various values of c_o at $t_o = 0$

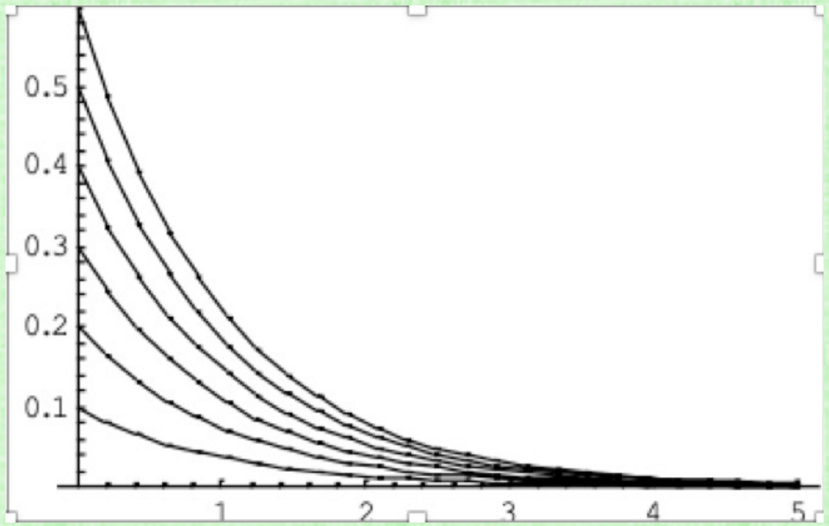
Gradient Flow

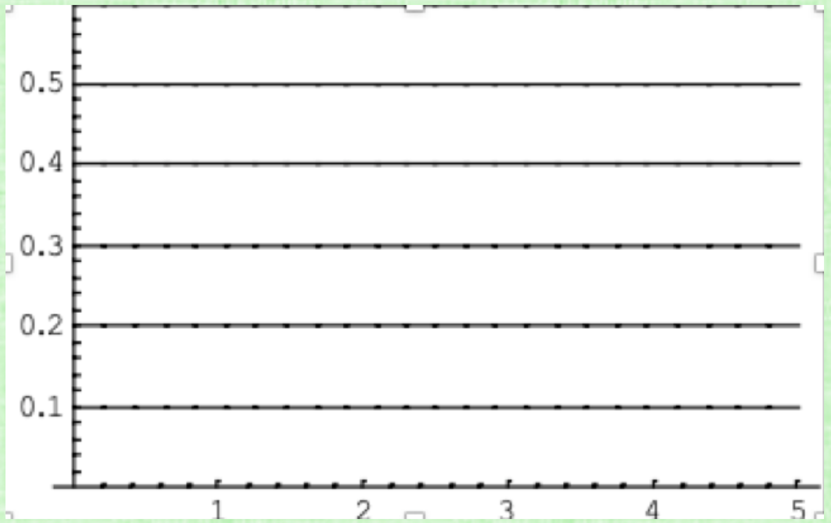
- Ansatz: following the solution trajectory in gradient flow leads to a preferred minimum of $\hat{f}(c)$
- Numerical errors cause perturbations away from this desired trajectory, and on to nearby trajectories (perhaps in the same family of solutions)
  - Hopefully, the perturbed trajectories stay close to the desired trajectory
  - Hopefully, the perturbed trajectories lead to the same minima
- Sometimes, there are <u>bifurcations</u> of solution trajectories
- In such regions, perturbations can lead to very different (presumably less preferred) minima

Posedness

- Consider $c' = \lambda c$ with solution family $c(t) = c_o e^{\lambda(t-t_o)}$







- $\lambda > 0$, exponential growth, <u>ill-posed</u>
- Small changes in initial conditions (and small solver errors) result in large changes to the trajectory

- $\lambda < 0$, exponential decay, <u>well-posed</u>
- Small changes in initial conditions (and small solver errors) are damped by converging trajectories

- $\lambda = 0$, constant solution, linearly stable, <u>mildly ill-posed</u>
- Small changes in initial conditions (and small solver errors) result in (slow, but cumulative) trajectory drift

Posedness for Systems

- A system of ODEs $c' = F(t,c)$ has a Jacobian matrix $J_F(t,c) = \frac{\partial F}{\partial c}(t,c)$
- Since $c(t)$ is time varying, so is $J_F(t,c(t))$
- Whenever an eigenvalue of $J_F(t,c(t))$ is positive, the associated part of the solution becomes ill-posed and trajectories can (wildly) diverge
  - This typically pollutes the entire solution vector
- Thus, <u>all eigenvalues</u> of $J_F(t,c(t))$ must be non-positive <u>for all</u> t under consideration for the problem to be considered well-posed
  - Moreover, eigenvalues close to zero may be suspect due to numerical errors
- Ill-posedness can rapidly lead to solution family bifurcation and thus minima far from what one might otherwise expect

Stability and Accuracy

- For a <u>well-posed</u> ODE, a numerical approach is considered <u>stable</u> if it does not overflow and produce NaNs (i.e. shoot off to an ∞ in parameter space)
- Stability is typically guaranteed via restrictions on the size of the time step Δt
  - Larger time steps lead to the method going <u>unstable</u>

- For a <u>well posed</u> ODE, a <u>stable</u> numerical approach can be analyzed for <u>accuracy</u> to see how well it matches known solutions

- Hopefully, stability and reasonable accuracy keep the numerical solution of the ODE close to an ideal trajectory (leading to the preferred minimum)

Forward Euler Method

- Approximate $c' = f(t, c)$ with $\frac{c^{q+1} - c^q}{\Delta t} = f(t^q, c^q)$
- Recursively: $c^{q+1} = c^q + \Delta t f(t^q, c^q)$

- Recall: Taylor series $c^{q+1} = c^q + \Delta t f(t^q, c^q) + O(\Delta t^2)$
- So, there is an $O(\Delta t^2)$ <u>local truncation error</u> each time step (i.e., each iteration)

- Since $\frac{t_f - t_o}{\Delta t} = O\left(\frac{1}{\Delta t}\right)$ time steps are taken, the total error or <u>global truncation error</u> is $O(\Delta t^2) O\left(\frac{1}{\Delta t}\right) = O(\Delta t)$
- Thus, the method is <u>1st order accurate</u>
    - Recall comments on accuracy and Newton-Cotes approaches in Unit 7 Curse of Dimensionality

Runge-Kutta (RK) Methods

- Taylor series can be used to (similarly) construct more accurate method:
- **<u>1st order:</u>** $\frac{c^{q+1}-c^q}{\Delta t} = f(t^q, c^q)$ which is the forward Euler method
- **<u>2nd order:</u>** $\frac{c^{q+1}-c^q}{\Delta t} = \frac{1}{2}k_1 + \frac{1}{2}k_2$ where $k_1 = f(t^q, c^q)$ is used in a forward Euler (predictor) update in order to compute $k_2 = f(t^{q+1}, c^q + \Delta t k_1)$
- **<u>4th order:</u>** $\frac{c^{q+1}-c^q}{\Delta t} = \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4$ where $k_1 = f(t^q, c^q)$, $k_2 = f\left(t^{q+\frac{1}{2}}, c^q + \frac{\Delta t}{2}k_1\right)$, $k_3 = f\left(t^{q+\frac{1}{2}}, c^q + \frac{\Delta t}{2}k_2\right)$, $k_4 = f(t^{q+1}, c^q + \Delta t k_3)$
  - Again, each term builds on the prior in a predictor style fashion

TVD Runge-Kutta Methods

- Combinations of forward Euler and averaging (since both are well-behaved)
- **1st order**: same as standard RK1 and forward Euler
- **2nd order:** same as standard RK2 (also called the midpoint rule, the modified Euler method, and Heun's predictor-corrector method)
  - Take two forward Euler steps: $\frac{\hat{c}^{q+1}-c^q}{\Delta t} = f(t^q, c^q)$ and $\frac{\hat{c}^{q+2}-\hat{c}^{q+1}}{\Delta t} = f(t^{q+1}, \hat{c}^{q+1})$
  - Then, average the initial and final state: $c^{q+1} = \frac{1}{2}c^q + \frac{1}{2}\hat{c}^{q+2}$
- **3rd order:** different from the standard RK3
  - Take two Euler steps, but average differently: $\hat{c}^{q+\frac{1}{2}} = \frac{3}{4}c^q + \frac{1}{4}\hat{c}^{q+2}$
  - Then, take another forward Euler step: $\frac{\hat{c}^{q+\frac{3}{2}}-\hat{c}^{q+\frac{1}{2}}}{\Delta t} = f\left(t^{q+\frac{1}{2}}, \hat{c}^{q+\frac{1}{2}}\right)$
  - Finally, average again: $c^{q+1} = \frac{1}{3}c^q + \frac{2}{3}\hat{c}^{q+\frac{3}{2}}$

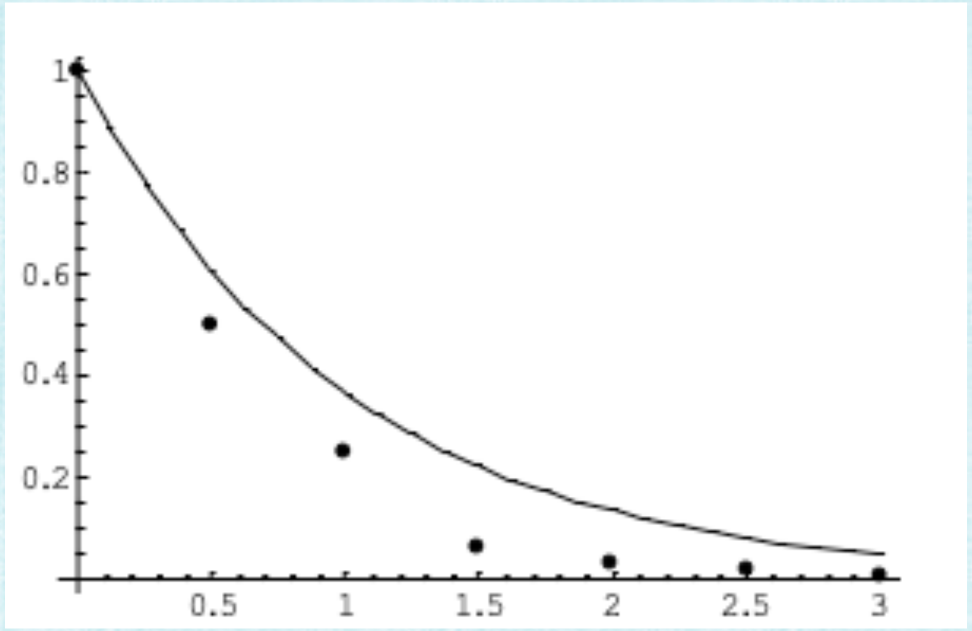
Stability Analysis

- Consider the model equation $c' = \lambda c$ with a well-posed $\lambda < 0$
  - This model equation is meant to illustrate how an eigenvalue λ of a Jacobian matrix might behave

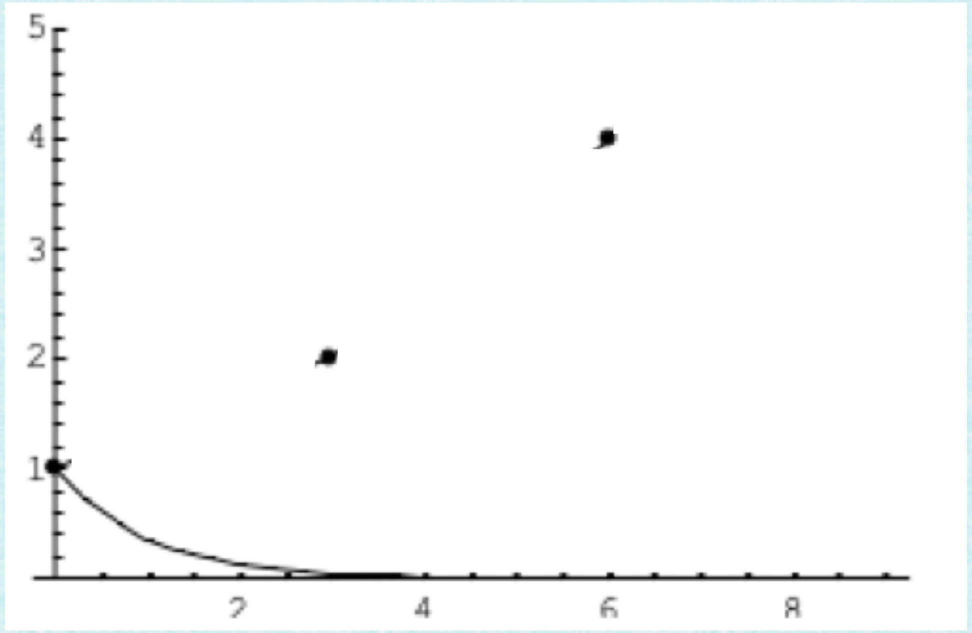
- Forward Euler gives $c^{q+1} = c^q + \Delta t \lambda c^q = (1 + \Delta t \lambda) c^q = (1 + \Delta t \lambda)^{q+1} c^o$
- The error shrinks and the solutions decays (as it should for $\lambda < 0$) as long as $|1 + \Delta t \lambda| < 1$
- This leads to $-1 < 1 + \Delta t \lambda < 1$ or $-2 < \Delta t \lambda < 0$ or $-\frac{2}{\lambda} > \Delta t > 0$
- Since $\lambda < 0$ and $\Delta t > 0$, one needs $\Delta t < \frac{2}{-\lambda}$ for stability
- This is called a time step restriction

Stability (an Example)

- Consider $c' = -c$ with $c(0) = 1$, where $\lambda = -1$ implies $\Delta t < 2$ for stability



- Here, $\Delta t = .5$ is stable
- Iterates (dots) track the solution (curve)



- Here, $\Delta t = 3$ is unstable
- Iterates (dots) grow exponentially
- The actual solution (curve) is shown decaying

Gradient Flow

- Using forward Euler on the gradient flow ODE gives: $c^{q+1} = c^q - \Delta t \nabla \hat{f}(c^q)$
- This is the exact same formula utilized for 1D line search $c^{q+1} = c^q + \Delta t \Delta c^q$ when using the steepest descent search direction $\Delta c^q = -\nabla \hat{f}(c^q)$

- Given this search direction, line search uses a 1D root/minimization approach to determine the next iterate
- This forward Euler interpretation suggests that one may instead choose Δt according to various ODE (or other similar) considerations

Adaptive Time Stepping

- ODEs utilize either a fixed size Δt or time varying Δt^q
    - The latter case is referred to as adaptive time stepping

- The ML community refer to Δt as the learning rate, and time steps as epochs

- When sub-iterations use only partially valid approximations of $-\nabla \hat{f}(c^q)$, e.g. mini-batch or SGD (unit 19), an epoch refers to one pass through the entire set of training data
    - i.e. each epoch allows the $-\nabla \hat{f}(c^q)$ estimates to see all the data

Adaptive Learning Rates

- Adagrad maintains a separate adaptive learning rate for each parameter, and modifies them based on past gradients computed for that parameter
  - Moving more/less in certain directions (because of per-parameter learning rates) changes the search direction
- Since the learning rates are based on a time history, the method is less localized and hopefully more robust (better behaved)

- Unfortunately, the learning rates monotonically decrease and often go to zero (stalling out the algorithm)
- Adadelta and RMSprop decrease the effect of prior gradients (similar in spirit to L-BFGS) so that the learning rate is not monotonically driven to zero

Implicit Methods

- Used to take larger time steps (compared to forward Euler and RK methods)
- Implicit methods have either no time step restriction or a very generous one

- However, one typically requires a nonlinear solver to advance each time step
- Sometimes, the nonlinear solver requires more computational effort than all the smaller (and simpler) time steps of forward Euler and/or RK combined (making it less efficient)

- The large time steps often lead to overly damped solutions (or unwanted oscillations)

Backward (Implicit) Euler

- $\frac{c^{q+1}-c^q}{\Delta t} = f(t^{q+1}, c^{q+1})$ is 1st order accurate with $O(\Delta t)$ error
- Stability: $\frac{c^{q+1}-c^q}{\Delta t} = \lambda c^{q+1}$ implies $c^{q+1} = \frac{1}{1-\Delta t\lambda} c^q$ where $0 < \left|\frac{1}{1-\Delta t\lambda}\right| < 1$
  - Thus, unconditionally stable since the inequality holds for all Δt (assuming $\lambda < 0$)
- Typically need to solve a nonlinear equation to find c^{q+1} (can be expensive)
- As $\Delta t \to \infty$, the method asymptotes to $f(t^{q+1}, c^{q+1}) = 0$, which is the correct steady state solution
  - But, overly damping makes one get there too fast, which is especially undesirable when the higher frequencies are important
- Great for stiff problems where high frequencies don't contribute much to the solution (and thus overly damping them is fine)

Implicit Stochastic Gradient Descent (ISGD)

- Used in Nonlinear Least Squares to overcome instabilities caused by using large time steps with forward Euler

- Forward Euler: $c^{q+1} = c^q - \Delta t \nabla \hat{f}(c^q)$
- Backward (implicit) Euler: $c^{q+1} = c^q - \Delta t \nabla \hat{f}(c^{q+1})$

- Since SGD only evaluates the gradient for one piece of data at a time, evaluating the gradient implicitly is a bit less unwieldy (as compared to doing so using all the data at the same time)

Trapezoidal Rule

- $\frac{c^{q+1}-c^q}{\Delta t} = \frac{f(t^q,c^q)+f\left(t^{q+1},c^{q+1}\right)}{2}$ is 2nd order accurate with $O(\Delta t^2)$ error
  - Averages forward Euler and backward Euler
- Stability: $\frac{c^{q+1}-c^q}{\Delta t} = \frac{\lambda c^q+\lambda c^{q+1}}{2}$ implies $c^{q+1} = \frac{1+\frac{\Delta t\lambda}{2}}{1-\frac{\Delta t\lambda}{2}} c^q$ where $0 < \left|\frac{1+\frac{\Delta t\lambda}{2}}{1-\frac{\Delta t\lambda}{2}}\right| < 1$
  - Thus, unconditionally stable since the inequality holds for all Δt (assuming $\lambda < 0$)
- Typically need to solve a nonlinear equation to find c^{q+1} (can be expensive)
- As $\Delta t \to \infty$, the method asymptotes to $f(t^{q+1}, c^{q+1}) = -f(t^q, c^q)$ which can cause unwanted oscillations
  - E.g., when $c' = \lambda c$, this is $c^{q+1} = -c^q$ which is oscillatory
  - More generally for $c' = f(t, c)$, this is $(c')^{q+1} = -(c')^q$ estimating the derivative as changing sign every iteration (causing oscillations)

Momentum

- Optimization methods often struggle when they are <u>too local</u>
- Adaptive learning rates based on <u>time history</u> (as discussed above) help to address this
- Momentum methods also aim to address this

- Momentum methods derive their motivation from <u>Newton's Second Law</u>
- Physical objects carry a <u>time history</u> of past interactions via their <u>momentum</u>
- The forces currently being applied to an object are combined with all previous forces to obtain the current trajectory/velocity

Newton’s Second Law

- Kinematics describe position $X(t)$, velocity $V(t)$, acceleration $A(t)$ as functions of time t via $\frac{dX}{dt}(t) = V(t)$ and $\frac{dV}{dt}(t) = A(t)$
  - Gradient flow $\frac{dc}{dt}(t) = -\nabla\hat{f}(c(t))$ is a kinematic equation
- Dynamics describe responses to external forces
  - Newton’s second law $F(t) = MA(t)$ is a dynamics equation
  - $V'(t) = A(t) = \frac{F(t)}{M}$ or $\frac{d^2X}{dt^2}(t) = X''(t) = \frac{F(t)}{M}$
- Combining kinematics and dynamics gives: $\begin{pmatrix} X'(t) \\ V'(t) \end{pmatrix} = \begin{pmatrix} V(t) \\ \frac{F(t,X(t),V(t))}{M} \end{pmatrix}$

Aside: First Order Systems

- Higher order ODEs are often reduced to first order systems
  - E.g. consider: $c'''' = f(t, c, c', c'', c''')$
  - Define new variables: $c_1 = c$, $c_2 = c'$, $c_3 = c''$, and $c_4 = c'''$
  - Then $\begin{pmatrix} c_1{}' \\ c_2{}' \\ c_3{}' \\ c_4{}' \end{pmatrix} = \begin{pmatrix} c_2 \\ c_3 \\ c_4 \\ f(t, c_1, c_2, c_3, c_4) \end{pmatrix}$ is an equivalent first order system

- Newton's second law $F = MX''$ can be written as $\begin{pmatrix} X' \\ V' \end{pmatrix} = \begin{pmatrix} V \\ F/M \end{pmatrix}$

Momentum Methods

- Newton's second law: $\begin{pmatrix} X'(t) \\ MV'(t) \end{pmatrix} = \begin{pmatrix} V(t) \\ F(t, X(t), V(t)) \end{pmatrix}$
  - The second equation augments the momentum with the current forces
  - That momentum is used in the first equation (after dividing by mass to get a velocity)

- Interpreting this from an optimization standpoint:
  - Instead of always using the current search direction, one should still be incorporating the effects of prior search directions
- This makes the optimization method less localized, and hopefully more robust (better behaved)

(Momentum-Style) Gradient Flow

- Split the forward Euler discretization $c^{q+1} = c^q - \Delta t \nabla \hat{f}(c^q)$ into two parts:

$$c^{q+1} = c^q + \Delta t v^q \quad \text{and} \quad v^q = -\nabla \hat{f}(c^q)$$

- Here, v^q is a velocity in parameter space

- Instead of setting the velocity equal to the (negative) gradient, treat gradients as forces that affect the velocity:

$$v^{q+1} = v^q - \Delta t \nabla \hat{f}(c^q)$$

- This results in a forward Euler discretization of $\begin{pmatrix} c'(t) \\ v'(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ -\nabla \hat{f}(c^q) \end{pmatrix}$

“The” ML Momentum Method

- The <u>original</u> momentum method is backward Euler on c and forward Euler on v, i.e. $c^{q+1} = c^q + \Delta t v^{q+1}$ and $v^{q+1} = v^q - \Delta t \nabla \hat{f}(c^q)$
  - Since the second equation can be updated first, the first equation doesn’t require a special solver
- Combining these into a single equation: $c^{q+1} = c^q + \Delta t v^q - \Delta t^2 \nabla \hat{f}(c^q)$
- Taking liberties to treat Δt and Δt^2 as two separate independent parameters leads to: $c^{q+1} = c^q + \alpha v^q - \beta \nabla \hat{f}(c^q)$
- Setting $\beta = \Delta t$ recovers the original discretization of gradient flow augmented with a new history dependent velocity term: $c^{q+1} = c^q + \alpha v^q - \Delta t \nabla \hat{f}(c^q)$
  - Writing this final equation as $c^{q+1} = c^q + \Delta t v^{q+1}$ illustrates an <u>inconsistent</u> velocity update of $v^{q+1} = \frac{\alpha}{\Delta t} v^q - \nabla \hat{f}(c^q)$

Nesterov Momentum

- Uses a predictor-corrector approach similar to 2nd order Runge- Kutta
- First, a forward Euler predictor step is taken $\hat{c}^{q+1} = c^q + \Delta t \hat{v}^{q+1}$ using a velocity of $\hat{v}^{q+1} = \frac{\alpha}{\Delta t} v^q$ (instead of $v^{q+1} = \frac{\alpha}{\Delta t} v^q - \nabla \hat{f}(c^q)$ from the last slide)
  - The current gradient information is ignored in the predictor step
  - Simplifying, the predictor step is $\hat{c}^{q+1} = c^q + \alpha v^q$
- Then, the gradient is evaluated at this new location $\hat{c}^{q+1}$ and used in "The" ML Momentum method: $c^{q+1} = c^q + \Delta t v^{q+1}$ and $v^{q+1} = \frac{\alpha}{\Delta t} v^q - \nabla \hat{f}(\hat{c}^{q+1})$
  - As a single equation: $c^{q+1} = c^q + \alpha v^q - \Delta t \nabla \hat{f}(\hat{c}^{q+1})$
  - Once again, there is an <u>inconsistent</u> velocity update $v^{q+1} = \frac{\alpha}{\Delta t} v^q - \nabla \hat{f}(\hat{c}^{q+1})$

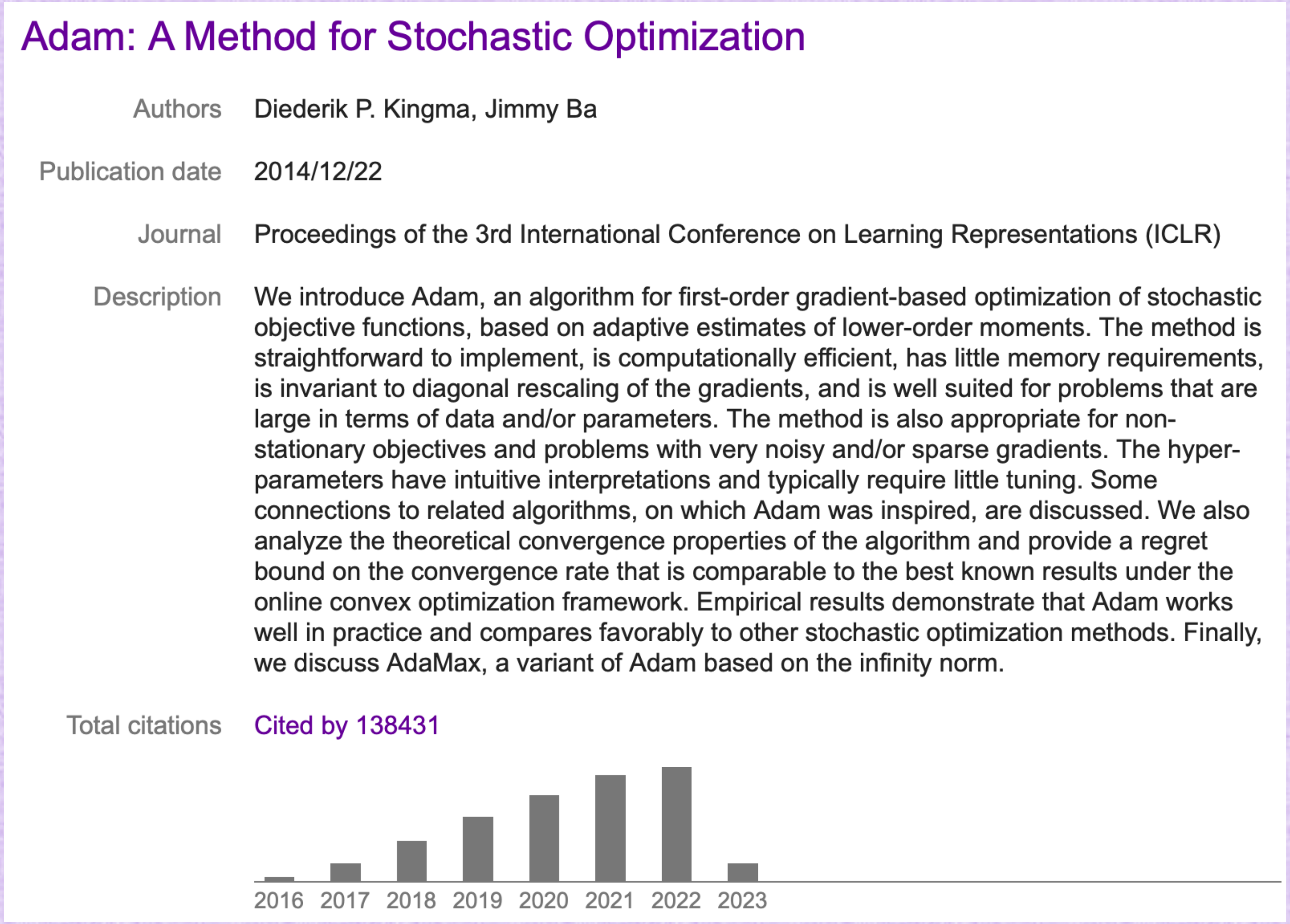
Physics/ODE Consistency

- Numerical ODE theory dictates (via consistency with the Taylor expansion) that the correct solution/path should be obtained as $\Delta t \to 0$
  - $c^{q+1} = c^q + \Delta t v^{q+1}$ properly resolves $c' = v$
  - But, $v^{q+1} = \frac{\alpha}{\Delta t} v^q - \nabla \hat{f}(\tilde{c})$ (with $\tilde{c}$ either c^q or $\hat{c}^{q+1}$) is problematic
- Revert to where we took liberties with $c^{q+1} = c^q + \alpha v^q - \beta \nabla \hat{f}(\tilde{c})$
- Choose $\beta = \hat{\beta} \Delta t^2$ (instead of $\beta = \Delta t$) to obtain $v^{q+1} = \frac{\alpha}{\Delta t} v^q - \Delta t \hat{\beta} \nabla \hat{f}(\tilde{c})$
- Setting $\alpha = \Delta t$ leads to a consistent $v^{q+1} = v^q - \Delta t \hat{\beta} \nabla \hat{f}(\tilde{c})$ where $\hat{\beta} > 0$ determines the strength of the steepest descent force
  - Forces (in physical systems) should be independent of Δt, and should accumulate to the same $O(1)$ net effect in $O(1)$ time (regardless of Δt)

Adam

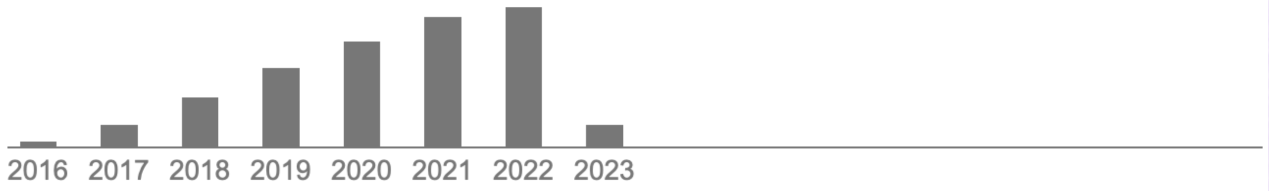
- Mixes ideas from adaptive learning rates and momentum methods:
  - Adaptive learning rate for each parameter (uses squared gradients to scale the learning rate, like RMSprop)
  - Uses a moving average of the gradient, like momentum methods
- AdaMax variant uses the L^∞ norm instead of the L^2 norm
- Nadam variant uses Nesterov momentum for the moving averages

- The original Adam paper had impressive results, which were duplicated by others, and the method has been quite popular
- Some recent work states that Adam might converge quicker than SGD w/momentum, but sometimes quicker to a worse solution (and so some practitioners are going back to SGD)
  - Still a lot to do!



Adam: A Method for Stochastic Optimization

Authors	Diederik P. Kingma, Jimmy Ba
Publication date	2014/12/22
Journal	Proceedings of the 3rd International Conference on Learning Representations (ICLR)
Description	We introduce Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. The method is straightforward to implement, is computationally efficient, has little memory requirements, is invariant to diagonal rescaling of the gradients, and is well suited for problems that are large in terms of data and/or parameters. The method is also appropriate for non-stationary objectives and problems with very noisy and/or sparse gradients. The hyper-parameters have intuitive interpretations and typically require little tuning. Some connections to related algorithms, on which Adam was inspired, are discussed. We also analyze the theoretical convergence properties of the algorithm and provide a regret bound on the convergence rate that is comparable to the best known results under the online convex optimization framework. Empirical results demonstrate that Adam works well in practice and compares favorably to other stochastic optimization methods. Finally, we discuss AdaMax, a variant of Adam based on the infinity norm.
Total citations	Cited by 138431



Constant Acceleration Equations

- Taylor expansion: $X^{q+1} = X^q + \Delta t V^q + \frac{\Delta t^2}{2} A^q + O(\Delta t^3)$
- In order to determine X^{q+1} with $O(\Delta t^3)$ accuracy, one only needs V^q with $O(\Delta t^2)$ accuracy and A^q with $O(\Delta t)$ accuracy
- In the system of equations for Newtons second law, $V' = F/M$ requires $O(\Delta t)$ less accuracy than $X' = V$ requires
- The standard kinematic formulas in basic physics use:
  - piecewise constant accelerations A^q
  - piecewise linear velocities $V^{q+1} = V^q + \Delta t A^q$
  - piecewise quadratic positions $X^{q+1} = X^q + \Delta t V^q + \frac{\Delta t^2}{2} A^q$

Newmark Methods

- $X^{q+1} = X^q + \Delta t V^q + \frac{\Delta t^2}{2}\left((1-2\beta)A^q + 2\beta A^{q+1}\right)$
- $V^{q+1} = V^q + \Delta t\left((1-\gamma)A^q + \gamma A^{q+1}\right)$
- $\beta = \gamma = 0$ constant acceleration equations (on the last slide)
- Second order accurate if and only if $\gamma = \frac{1}{2}$, i.e. $V^{q+1} = V^q + \Delta t \frac{A^q + A^{q+1}}{2}$
- $\gamma = \frac{1}{2}, \beta = \frac{1}{4}$ is Trapezoidal Rule (on both X and V)
  - $X^{q+1} = X^q + \Delta t V^q + \frac{\Delta t^2}{4}(A^q + A^{q+1})$ becomes $X^{q+1} = X^q + \Delta t V^q + \frac{\Delta t}{2}(V^{q+1} - V^q)$ or $X^{q+1} = X^q + \Delta t \frac{V^q + V^{q+1}}{2}$
- $\gamma = \frac{1}{2}, \beta = 0$ is Central Differencing: $X^{q+1} = X^q + \Delta t V^q + \frac{\Delta t^2}{2} A^q$

Central Differencing

- $X^{q+1} = X^q + \Delta t V^q + \frac{\Delta t^2}{2} A^q$ and $V^{q+1} = V^q + \Delta t \frac{A^q + A^{q+1}}{2}$
- Adding $X^{q+2} = X^{q+1} + \Delta t V^{q+1} + \frac{\Delta t^2}{2} A^{q+1}$ to $X^{q+1} = X^q + \Delta t V^q + \frac{\Delta t^2}{2} A^q$ gives $X^{q+2} - X^q = \Delta t(V^q + V^{q+1}) + \frac{\Delta t^2}{2}{}^2 (A^q + A^{q+1}) = \Delta t(V^q + V^{q+1}) + \Delta t(V^{q+1} - V^q) = 2\Delta t V^{q+1}$
- So $V^{q+1} = \frac{X^{q+2} - X^q}{2\Delta t}$ (a second order accurate central difference)
- Subtracting (same equations) gives $X^{q+2} - 2X^{q+1} + X^q = \Delta t(V^{q+1} - V^q) + \frac{\Delta t^2}{2}(A^{q+1} - A^q) = \frac{\Delta t^2}{2}(A^q + A^{q+1}) + \frac{\Delta t^2}{2}(A^{q+1} - A^q) = \Delta \mathrm{t}^2 \mathrm{A}^{q+1}$
- So $A^{q+1} = \frac{X^{q+2} - 2X^{q+1} + X^q}{\Delta t^2}$ (a second order accurate central difference)

Staggered Position and Velocity

- Update position with a staggered velocity $X^{q+1} = X^q + \Delta t V^{q+\frac{1}{2}}$
- Using averaging $V^{q+1} = \frac{V^{q+\frac{1}{2}}+V^{q+\frac{3}{2}}}{2}$ which still equals $\frac{X^{q+2}-X^q}{2\Delta t}$ as desired
- $A^{q+1} = \frac{(X^{q+2}-X^{q+1})-(X^{q+1}-X^q)}{\Delta t^2} = \frac{V^{q+\frac{3}{2}}-V^{q+\frac{1}{2}}}{\Delta t}$
- This last term is equal to both $\frac{V^{q+1}-V^{q+\frac{1}{2}}}{(\Delta t/2)}$ and $\frac{V^{q+\frac{3}{2}}-V^{q+1}}{(\Delta t/2)}$
- So $V^{q+1} = V^{q+\frac{1}{2}} + \frac{\Delta t}{2} A^{q+1}$ and $V^{q+\frac{3}{2}} = V^{q+1} + \frac{\Delta t}{2} A^{q+1}$
- The second equation shifted one index is $V^{q+\frac{1}{2}} = V^q + \frac{\Delta t}{2} A^q$

Staggered Central Differencing

- $V^{q+\frac{1}{2}} = V^q + \frac{\Delta t}{2} A(X^q, V^q)$ and $X^{q+1} = X^q + \Delta t V^{q+\frac{1}{2}}$ are explicit
- $V^{q+1} = V^{q+\frac{1}{2}} + \frac{\Delta t}{2} A(X^{q+1}, V^{q+1})$ is explicit in X but implicit in V
- Position based forces (e.g. elasticity) are typically nonlinear making them hard to invert (good that we don't have to), whereas velocity based forces (e.g. damping) are typically linear making them easier to invert (which we need to)
- Position based forces are often important for material behavior (good we don't overdamp them), whereas velocity based damping doesn't suffer much from increased damping (which we do if we switch from trapezoidal rule to backward Euler in the last step, i.e. $V^{q+1} = V^q + \Delta t A(X^{q+1}, V^{q+1})$)
- Position based forces don't require too stringent a time step restriction (good, because we need one), whereas velocity based forces typically require a very small time step restriction (which we can ignore with an implicit solve)