AI & Robotics: Research

Differentiable AI

Marc Toussaint
Technical University of Berlin
Summer 2020

Outline

- Motivation
- Computation Graphs & Chain Rules
- Neural Networks
- End-to-end training: Embedding other computations:
  - Differentiable Optimization
  - Differentiable Physics
  - Value Iteration Networks
- Discussion:
  - Are gradients all we need?

Motivation

- Why is gradient-based optimization so omnipresent now?
  - Collateral effect of neural networks!
  - NN development lead to great creativity in designing architectures
  - $\rightarrow$ great creativity in embedding more that just neural layers

Motivation

- Why is gradient-based optimization so omnipresent now?
  - Collateral effect of neural networks!
  - NN development lead to great creativity in designing architectures
  - $\rightarrow$ great creativity in embedding more that just neural layers

- Perhaps it provides a general(?) “language” for AI system design
  - something that previous “cognitive architectures” failed with
  - something that software engineering does not give satisfying answers
  - something that is fundamentally hard in robotics

Motivation

- Why is gradient-based optimization so omnipresent now?
  - Collateral effect of neural networks!
  - NN development lead to great creativity in designing architectures

  $\rightarrow$ great creativity in embedding more that just neural layers

- Perhaps it provides a general(?) “language” for AI system design
  - something that previous “cognitive architectures” failed with
  - something that software engineering does not give satisfying answers
  - something that is fundamentally hard in robotics

    (But are computation graphs really the best language?)

- Other examples for “general architecture languages”:
  - graphical models (one computational principle: probabilistic inference)
  - cognitive architectures, alternative neural architectures (e.g., fields)
  - but none of them very successful for large scale systems

Computation Graphs

(more details: `Lecture-Maths.pdf`)

Partial derivative

- Given a function $f:\ \mathbb{R}^n \to \mathbb{R}$ of n arguments, $f(x_1, .., x_n)$, the *partial* derivative is the standard derivative w.r.t. only one of its arguments:

$$\frac{\partial}{\partial x_i} f(x_1, .., x_n) = \lim_{h \to 0} \frac{f(x_1, .., x_i + h, .., x_n) - f(x)}{h} \ .$$

Computation Graphs

- A **computation graph** (or **function network**) is a DAG of n variables x_i where each variable is a deterministic function of a set of parents $\pi(i) \subset \{1, .., n\}$, that is
$$x_i = f_i(x_{\pi(i)})$$
where $x_{\pi(i)} = (x_j)_{j \in \pi(i)}$ is the tuple of parent values
- (This could also be called *deterministic Bayes net*.)

Computation Graphs

- A **computation graph** (or **function network**) is a DAG of n variables x_i where each variable is a deterministic function of a set of parents $\pi(i) \subset \{1, .., n\}$, that is

$$x_i = f_i(x_{\pi(i)})$$

  where $x_{\pi(i)} = (x_j)_{j \in \pi(i)}$ is the tuple of parent values
- (This could also be called *deterministic Bayes net*.)

- Given a computation graph, we can define the **total derivative**: Given a variation dx of some variable, how would another variable vary, accounting for all dependencies down the DAG, in the linear limit?

Example

$$f(x,g) = 3x + 2g \quad \text{and} \quad g(x) = 2x$$

What is the "derivative of f w.r.t. x"?

Example

$$f(x, g) = 3x + 2g \quad \text{and} \quad g(x) = 2x$$

What is the "derivative of f w.r.t. x"?

$$\frac{\partial}{\partial x} f(x, g) = 3$$

Example

$$f(x, g) = 3x + 2g \quad \text{and} \quad g(x) = 2x$$

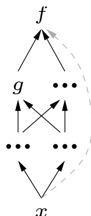
What is the "derivative of f w.r.t. x"?

$$\frac{\partial}{\partial x} f(x, g) = 3$$

$$\frac{df}{dx} = \frac{\partial}{\partial x}\Big[3x + 2(2x)\Big] = 7$$

$$\frac{df}{dx} = \frac{\partial}{\partial x} f(x, g) + \frac{\partial}{\partial g} f(x, g)\ \frac{\partial}{\partial x} g(x) = 3 + 2 \cdot 2 = 7$$

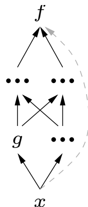
Chain rules



- Forward-version: (I use for robotics Jacobians)

$$\frac{df}{dx} = \sum_{g \in \pi(f)} \frac{\partial f}{\partial g} \frac{dg}{dx}$$

Why "forward"? You've computed $\frac{dg}{dx}$ already, now you move forward to $\frac{df}{dx}$.
Note: If $x \in \pi(f)$ is also a direct argument to f, the sum includes the term $\frac{\partial f}{\partial x} \frac{dx}{dx} \equiv \frac{\partial f}{\partial x}$.
To emphasize this, one could also write $\frac{df}{dx} = \frac{\partial f}{\partial x} + \sum_{\substack{g \in \pi(f) \\ g \neq x}} \frac{\partial f}{\partial g} \frac{dg}{dx}$.



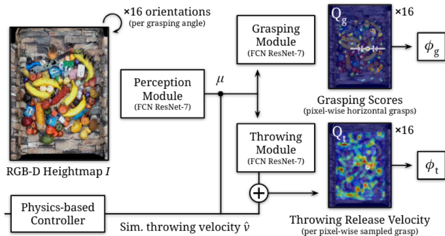
- Backward-version: (used in NNs)

$$\frac{df}{dx} = \sum_{g : x \in \pi(g)} \frac{df}{dg} \frac{\partial g}{\partial x}$$

Why "backward"? You've computed $\frac{df}{dg}$ already, now you move backward to $\frac{df}{dx}$.
Note: If $f \in \pi(g)$, the sum includes $\frac{df}{df} \frac{\partial f}{\partial x} \equiv \frac{\partial f}{\partial x}$. We could also write
$\frac{df}{dx} = \frac{\partial f}{\partial x} + \sum_{\substack{g : x \in \pi(g) \\ g \neq f}} \frac{df}{dg} \frac{\partial g}{\partial x}$.

Example: TossingBot "Physics-based Controller"

- Figure 4 of the Tossingbot paper displays the computation graph:



- Output error gradients are defined by
  - A discriminative loss for ϕ_g, where the executed grasp decision (single pixel) is labelled 1 if throwing succeeded, and 0 otherwise
  - A regression (Huber) loss for ϕ_t to minimize $|\delta - \bar{\delta}|$, where $\bar{\delta} = v - \hat{v}(\bar{p})$ is the target residual if $\bar{p}$ would have been the goal (added to buffer)
  - Backprop gives $\frac{d\mathcal{L}}{d\hat{v}}$
- The physics-based velocity is $\hat{v} = \sqrt{\frac{a(p_x^2+p_y^2)}{(r_z-p_z-\sqrt{p_x^2+p_y^2})}}$

  Which gives the partial, $\frac{\partial \hat{v}}{\partial a} = \frac{1}{2\sqrt{a}} \cdots$, and total derivative $\frac{d\mathcal{L}}{da} = \frac{d\mathcal{L}}{d\hat{v}} \frac{\partial \hat{v}}{\partial a}$.

Neural Networks

(more details in `Lecture-MachineLearning.pdf`)

Basic feed-forward Neural Network

- L-layer feed-forward NN is parameterized by
  - layer sizes $h = (h_0, .., h_L)$, with h_0: input dims, h_L: output dims, $L - 1$ hidden layers
  - weights $W_l \in \mathbb{R}^{h_l \times h_{l\text{-}1}}$ and biases $b_l \in \mathbb{R}^{h_l}$

Basic feed-forward Neural Network

- L-layer feed-forward NN is parameterized by
  - layer sizes $h = (h_0, .., h_L)$, with h_0: input dims, h_L: output dims, $L - 1$ hidden layers
  - weights $W_l \in \mathbb{R}^{h_l \times h_{l\text{-}1}}$ and biases $b_l \in \mathbb{R}^{h_l}$

- The forward mapping $\mathbb{R}^{h_0} \mapsto \mathbb{R}^{h_L}$ iteratively computes:
  - the **input** to layer l is $z_l = W_l x_{l\text{-}1} + b_l \in \mathbb{R}^{h_l}$ *(linear)*
  - the **activation** of layer l is $x_l = \sigma(z_l) \in \mathbb{R}^{h_l}$ *(non-linear)*

  except for the last layer, where z_L is the network output (without non-linearity).

Applying the Backward Chain Rule

- Forward equations:

$$z_l = W_l x_{l\text{-}1} + b_l$$
$$x_l = \sigma(z_l)$$

Applying the Backward Chain Rule

- Forward equations:

$$z_l = W_l x_{l\text{-}1} + b_l$$
$$x_l = \sigma(z_l)$$

- Given $\delta_L \triangleq \frac{d\mathcal{L}}{dz_L} \in \mathbb{R}^{1\times M}$ as the gradient (as row vector) at the output, we have

$$\forall_{l=L\text{-}1,..,1}: \ \delta_l \triangleq \frac{d\mathcal{L}}{dz_l} = \frac{d\mathcal{L}}{dz_{l+1}} \frac{\partial z_{l+1}}{\partial x_l} \frac{\partial x_l}{\partial z_l} = [\delta_{l+1}\ W_{l+1}] \circ [\sigma'(z_l)]^\top$$

where $\circ$ is an *element-wise product*

- The gradient w.r.t. parameters is:

$$\frac{d\mathcal{L}}{dW_{l,ij}} = \frac{d\mathcal{L}}{dz_{l,i}} \frac{\partial z_{l,i}}{\partial W_{l,ij}} = \delta_{l,i}\ x_{l\text{-}1,j} \quad \text{or} \quad \frac{d\mathcal{L}}{dW_l} = \delta_l^\top x_{l\text{-}1}^\top\ , \quad \frac{d\mathcal{L}}{db_l} = \delta_l^\top$$

Convolutional NNs

- In every layer we have $x_l \in \mathbb{R}^{h_l \times w_l \times c_l}$ activations
- Each activation x_l is a linear function of only some of the activations in $x_{l\text{-}1}$, which is called the *receptive field* with filter size f_l
- The lth layer has $f_l^2 c_l c_{l\text{-}1}$ parameters

- The stride and padding determine the size of the next layer:
  - If there is no padding, a stride S reduces width by $w_l = (w_{l\text{-}1} - f_l)/S + 1$, which works if f_l is a multiple of S (e.g. max pooling).
  - One often pads the images with zeros to ensure that $w_l = w_{l\text{-}1}/S$, or $w_l = (w_{l\text{-}1} - 1)/S + 1$
  - If the image is padded with P zeros on all sides, we have $w_l = (w_{l\text{-}1} - f_l + 2P)/S + 1$

TossingBot example

- Perception network: C(3,64)-MP-RB(128)-MP-RB(256)-RB(512)
  - where C(k,c) denotes a convolutional layer with k$\times$k filters and c channels
  - RB(c) denotes a residual block [14] with two convolutional layers using 3$\times$3 filters and c channels
  - MP denotes a 3$\times$3 max pooling layer with stride = 2
- Sizes:
  - Input: height map 180 x 140 x 1
  - C(3,64): conv+relu layer of size 180 x 140 x 64, with $9 * 64 * 1$ parameters
  - MP: pooling layer of size 90 x 70 x 64, no parameters
  - RB(128): 2 layers (conv+relu+conv), each of size 90 x 70 x 128, first with $9 * 128 * 64$ parameters, 2nd with $9 * 128 * 128$ parameters
  - (followed by a relu)
  - MP: pooling layer of size 45 x 35 x 128, no parameters

  $\vdots$

End-to-end training: Embedding other computations

- Differentiable Optimization
- Differentiable Physics
- Value Iteration Networks

Differentiating complex solvers

- There are algorithms to solve all kinds of complex problems
- If each little step of the algorithm is differentiable, we can differentiate through the whole algorithm using the chain rule

- However, the more interesting case is if we have additional theory that tells us directly the derivative of outputs w.r.t. inputs

- **Sensitivity Analysis**: A classical field that analizes, in linear approximation, how sensitive outputs are w.r.t. inputs. That's exactly what we need!

Sensitivity Analysis in Optimization

- Consider a general problem

$$x^* = \underset{x}{\operatorname{argmin}}\, f(x) \quad \text{s.t.} \quad g(x) \leq 0,\ h(x) = 0$$

  – where $x \in \mathbb{R}^n,\ f:\ \mathbb{R}^n \to \mathbb{R},\ g:\ \mathbb{R}^n \to \mathbb{R}^m,\ h:\ \mathbb{R}^n \to \mathbb{R}^{m'}$, all smooth
  – First compute the optimum, potentially using many iterations
  – Then “differentiate” the optimum, with just a single equation

Sensitivity Analysis in Optimization

- Consider a general problem

$$x^* = \underset{x}{\operatorname{argmin}}\, f(x) \quad \text{s.t.} \quad g(x) \leq 0,\ h(x) = 0$$

  - where $x \in \mathbb{R}^n,\ f:\ \mathbb{R}^n \to \mathbb{R},\ g:\ \mathbb{R}^n \to \mathbb{R}^m,\ h:\ \mathbb{R}^n \to \mathbb{R}^{m'}$, all smooth
  - First compute the optimum, potentially using many iterations
  - Then "differentiate" the optimum, with just a single equation

- How would the output (optimum) differ with different inputs (if f, g, h vary)?

Sensitivity Analysis in Optimization

- KKT conditions: x optimal $\Rightarrow \exists \lambda \in \mathbb{R}^m, \nu \in \mathbb{R}^{m'}$ such that

$$\nabla f(x) + \nabla g(x)^\top \lambda + \nabla h(x)^\top \nu = 0 \quad (1)$$

$$g(x) \leq 0 \; , \quad h(x) = 0 \; , \quad \lambda \geq 0 \quad (2)$$

$$\lambda \circ g(x) = 0 \; , \quad (3)$$

- Consider infinitesimal variation $\tilde{f} = f + \epsilon \hat{f}$, $\tilde{g} = g + \epsilon \hat{g}$, $\tilde{h} = h + \epsilon \hat{h}$; how does x^* vary?
  - The KKT resitual will be

$$\hat{r} = \begin{pmatrix} \nabla \hat{f} + \nabla \hat{g}^\top \lambda + \nabla \hat{h}^\top \nu \\ \hat{h} \\ \lambda \circ \hat{g} \end{pmatrix}$$

  - The primal-dual Newton step will be

$$\begin{pmatrix} \hat{x} \\ \hat{\lambda} \\ \hat{\nu} \end{pmatrix} = - \begin{pmatrix} \nabla^2 f & \nabla g^\top & \nabla h^\top \\ \nabla h & 0 & 0 \\ \text{diag}(\lambda) \nabla g & \text{diag}(g) & 0 \end{pmatrix}^{\text{-1}} \begin{pmatrix} \nabla \hat{f} + \nabla \hat{g}^\top \lambda + \nabla \hat{h}^\top \nu \\ \hat{h} \\ \lambda \circ \hat{g} \end{pmatrix}$$

- The new optimum is at $x^* + \hat{x}$
  - Insight: This derivation assumes stability of constraint activity, which is "standard constraint qualification" in the optimization literature

Classical literature on differentiation through NLP solutions

- Ralph & Dempe. **Directional derivatives of the solution of a parametric nonlinear program. 1994**. Research Report.
- Fiacco & Kyparisis. **Sensitivity analysis in nonlinear programming** under second order assumptions. Lecture Notes in Control and Information Sciences, 74-97, **1985**.
- Kyparisis. Sensitivity analysis for nonlinear programs and variational inequalities with nonunique multipliers. Mathematics of Operations Research, 15:286298, 1990.
- Levy & Rockafellar. Sensitivity analysis of solutions to generalized equations. Trans. Amer. Math. Soc. 1993.
- Poliquin & Rockafellar. **Proto-derivative** formulas for basic **subgradient mappings** in mathematical programming. Set-valued Analysis, 2:275290, 1994.
- Levy & Rockafellar. Sensitivity of solutions in nonlinear programs with nonunique multiplier. Recent Adv. in Nonsmooth Optimzation: 215-223, 1995

Classical literature on differentiation through NLP solutions

- Ralph & Dempe. **Directional derivatives of the solution of a parametric nonlinear program. 1994**. Research Report.
- Fiacco & Kyparisis. **Sensitivity analysis in nonlinear programming** under second order assumptions. Lecture Notes in Control and Information Sciences, 74-97, **1985**.
- Kyparisis. Sensitivity analysis for nonlinear programs and variational inequalities with nonunique multipliers. Mathematics of Operations Research, 15:286298, 1990.
- Levy & Rockafellar. Sensitivity analysis of solutions to generalized equations. Trans. Amer. Math. Soc. 1993.
- Poliquin & Rockafellar. **Proto-derivative** formulas for basic **subgradient mappings** in mathematical programming. Set-valued Analysis, 2:275290, 1994.
- Levy & Rockafellar. Sensitivity of solutions in nonlinear programs with nonunique multiplier. Recent Adv. in Nonsmooth Optimzation: 215-223, 1995

"We show under a standard constraint qualification, not requiring uniqueness of the multipliers, that the quasi-solution mapping is differentiable in a generalized sense, and we present a formula for its derivative."

- Quasi-solution mapping: parameterized NLP $\mathcal{P}(\theta)$

$$S : \theta \mapsto \{x : \text{KKT hold for } \mathcal{P}(\theta)\}$$

Sensitivity Analysis in Optimization

- Bottom line: We can analyze how changes in the optimization problem translate to changes of the optimium x^*

- Differentiable Optimization
  - Can be embedded in auto-differentiation computation graphs (Tensorflow)
  - Important implications for Differentiable Physics
  - **But: Not differentiable across constraint activations**

Related Work

Agrawal et al: *Differentiable convex optimization layers*. NIPS'19

Making Robotics Algorithms Differentiable

- Many robotics problems can be casted as optimization
- Many robotics computations are differentiable (Jacobians)

- But the solvers and esp. planners not easily

- Two examples (also to highlight the issues):
  - Differentiable Physics
  - Value Iteration Networks

Differentiable Physics

- Todorov: A convex, smooth and invertible contact model for trajectory optimization. ICRA'11
- de Avila Belbute-Peres & Kolter: A Modular Differentiable [..] Physics Engine. NIPS'17 workshop
- Mordatch et al: Discovery of complex behaviors through contact-invariant optimization. TOG'12
- Toussaint et al: Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning. RSS'18

- A Physics Simulator iterates forward. Assume you have a code that depends on the end state of the physics simulator. Can you backprop gradients through the simulator? To compute the gradient of the cost w.r.t. any input to the simulator?

Differentiable Physics

- Todorov: A convex, smooth and invertible contact model for trajectory optimization. ICRA'11
- de Avila Belbute-Peres & Kolter: A Modular Differentiable [..] Physics Engine. NIPS'17 workshop
- Mordatch et al: Discovery of complex behaviors through contact-invariant optimization. TOG'12
- Toussaint et al: Differentiable Physics and Stable Modes for Tool-Use and Manipulation Planning. RSS'18

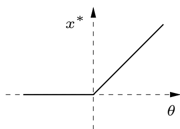
- A Physics Simulator iterates forward. Assume you have a code that depends on the end state of the physics simulator. Can you backprop gradients through the simulator? To compute the gradient of the cost w.r.t. any input to the simulator?
- In principle yes, **but only piece-wise differentiable! not through contacts!**

Non-differentiable Physics

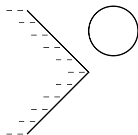
- Basic example $(x, \theta \in \mathbb{R})$:

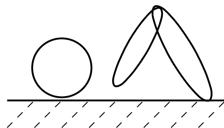
$$\min_x (x-\theta)^2 \quad \text{s.t.} \quad x \geq 0$$

$$S : \theta \mapsto x^* = \max\{0, x\}$$

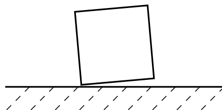


- Bifurcation depending on contact:





- Jumping contact points:



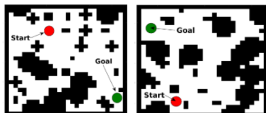
Chaotic system: tiny change in initial condition (θ), huge change in outcome

Value Iteration Networks

- Value Iteration in Markov Decision Processes: (initialize $V_{k=0}(s) = 0$)

$$\forall s: \; V_{k+1}(s) = \max_a \Big[R(s,a) + \gamma \sum_{s'} P(s'|s,a) \; V_k(s') \Big]$$

- Input: R and P Output: V_k for some fixed k
- You can think of $V_0, V_1, ..., V_k$ as layers of a network – easy to differentiate



- Tricky: How to encode P and R?
  - Value Iteration Network: Assume a 2D maze. P is given by a 2D maze image, with obstacles and goals having different color
  - The state space is equally 2D, V_k is a 2D activation
  - Take that image as the input to the value iterations

Tamar, Thomas, Levine, Abbeel: *Value iteration networks*. NIPS 16

Related work

- Must Read:

  Karkus, Ma, Hsu, Kaelbling, Lee, Lozano-Prez: *Differentiable Algorithm Networks for Composable Robot Learning*. RSS'19

  - Discusses the idea explicitly for robot architectures
  - Good overview on existing differentiable algorithms that can be used as components in a larger system
  - Interesting series of demonstrations that include more and more challenges on the vision and control side

Discussion

Disclaimer: Highly subjective!

Are gradients all we need?

Are gradients all we need?

- Gradients allow you to walk downhill
- Gradients do not help with local optima   (esp. in combinatorics)

  (In high-dimensions: saddle-point issues, see below.)

Are gradients all we need?

- Many problems in AI are inherently non-convex (i.e., have many local optima, often infeasible or non-satisficing local optima)
  - Physical Reasoning (own work)
  - Logic planning, planning in general
  - Scheduling, many optimization problems in operations research

- So why are gradients so successful right now?

Are gradients all we need?

- Many problems in AI are inherently non-convex (i.e., have many local optima, often infeasible or non-satisficing local optima)
  - Physical Reasoning (own work)
  - Logic planning, planning in general
  - Scheduling, many optimization problems in operations research

- So why are gradients so successful right now?

Solving the problems from scratch might be highly non-convex
But solving them using lots of example data might be less convex

- Some reasoning/control/decision problem might be highly non-convex
- But if you have lots of example data
- Training a non-linear system to mimic the data and thereby solve the reasoning/control/decision problem might be much less convex

- Arguably, the current success of ML empirically proves exactly that

- Training non-linear systems is still highly non-convex
  - But recent advances in NN theory give insight:

    Dauphin et al: *Identifying and attacking the saddle point problem in high dimensional non-convex optimization*. NIPS'13

    Kawaguchi: *Deep Learning without Poor Local Minima*. NIPS'16

    Choromanska: *The loss surfaces of multilayer networks*. Artificial intelligence and statistics. 2015

- Training non-linear systems to solve highly non-convex tasks – by mimicking example data – is "convex enough" for gradients to work well

Data Data Data

Accepting this view (although it might have limits)..

- It all boils down to data generation
  - Some of the major advances are about “self-generating” data: GANs, self-play
  - Tricks such as, hindsight supervised data (cf. TossingBot)

Data Data Data

Accepting this view (although it might have limits)..

- It all boils down to data generation
  - Some of the major advances are about “self-generating” data: GANs, self-play
  - Tricks such as, hindsight supervised data (cf. TossingBot)
  - Need classical solvers (or humans) to generate data?
  - Need models and simulators to generate data?