DeePC HUNT
Data-enabled Predictive Control Hyperparameter Tuning via Differentiable Optimization Layers

Michael Cummins - 22-908-255
Supervisors : Alberto Padoan, Keith Moffat

July 20, 2023

1 Abstract

This project explores the problem of automatic tuning of direct data-driven control algorithms in control system design. While traditional control design approaches rely on system identification followed by model-based control, recent advancements in machine learning, increased computing power, and the availability of large data sets have sparked renewed interest in direct approaches that infer optimal decisions directly from measured data. In particular, the direct data-driven predictive control (DeePC) algorithm is particularly effective for nonlinear and uncertain systems. However, achieving satisfactory control performance using this algorithm requires careful selection of hyperparameters, which is typically done manually through intuition and trial-and-error. In this paper, we propose a backpropagation-based method for hyperparameter optimization, which outperforms manual tuning strategies. Experimental results demonstrate the effectiveness of our approach.

2 Introduction

DeePC is a powerful control policy which has outperformed MPC on selected tasks [7], [8]. However, the performance of DeePC on nonlinear and stochastic systems is highly sensitive to the choice of regularization parameters [10], which are presently chosen heuristically. This can be difficult, particularly when there is a strong coupling between parameters or when you instantiate multiple regularizations. We demonstrate the effectiveness of a local search method for the optimal hyperparameters via a projected gradient descent method [3].

In Sections 3 and 4, we outline the related work based around differentiable optimization layers and necessary preliminaries for our method. In section 5, we described the formulation of the problem specific to two systems and present results that validate our approach. conclusions and future work are discussed in section 6.

3 Related Work

3.1 Differentiable-MPC

The method we our proposing is essentially a model-free variant of differentiable MPC [6]. However, there are a few significant differences. Differentiable MPC is instantiated as a quadratic program and the gradients of the optimal trajectory $\tau^*_{1:T}$ with respect to the problem data $\{C_t, c_t, F_t, f_t\}_{1:T}$ are obtained by differentiating the KKT conditions, as described in [5] and [6]. For example, given an MPC problem of the form;

$$\arg\min_{\tau_{1:T}} \sum_{t=1}^{T} \frac{1}{2}\ \tau_t^\intercal C_t \tau_t + c_t^\intercal \tau_t \ \text{ subject to } \ x_1 = x_{\text{init}}, x_{t+1} = F_t\tau_t + f_t \tag{1}$$

Where $\tau^*_{1:T} = \{x^*_t, u^*_t\}_{1:T}, (x^*_t, u^*_t) \in \mathbb{R}^{m+n}$, we would like to use differentiable optimization to tune one of our policy parameters $\{C_t, c_t, F_t, f_t\}_{1:T}$. Due to the differentiable nature of the policy, we can perform updates on these parameters in an end-to-end fashion using differentiable optimization. An example is illustrated in (2), starting from an initial guess $C_t^{(0)}$ and using step size η;

$$C_t^{(k+1)} = C_t^{(k)} - \eta \nabla_{C_t} \mathcal{L}(\tau^*(C, c, F, f)) \tag{2}$$

Where $\nabla_{C_t}\tau^*(C, c, F, f)$ is obtained through differentiating the KKT conditions of (1) and $\mathcal{L}$ is a generic loss function defined as $\mathcal{L} : \mathbb{R}^{(m+n)T} \to \mathbb{R}$. We would like to use this technique in the context of DeePC, to learn the regularization parameters, which will be introduced later. However, this technique is restricted to a standard form (1) which we want to avoid since this can be difficult to formulate when introducing regularizations and slack variables.

3.2 Differentiable Convex Optimization Layers

Differentiable convex optimization layers addresses the issue of formulating your optimization problem in standard form by utilizing cvxpy [9]. When using cvxpy, we can instantiate our convex optimization problem at a high level in the form of parameters, variables and constants denoted by Ψ. Once instantiated, our problem is then transformed into a cone program (1) through a sparse linear projection C, where $C(\Psi) = (A, b, c)$, $(A, b, c) \in \mathbb{R}^{m\times n} \times \mathbb{R}^m \times \mathbb{R}^n$, $\mathcal{K}$ is a nonempty, closed, convex cone and x is the decision variable of the original high level optimization problem.

$$\min_{x,\nu} c^\intercal x \ \text{ subject to } \ Ax + \nu = b,\ (x, \nu) \in \mathbb{R}^n \times \mathcal{K} \tag{3}$$

The solution to original high level convex optimization problem is then simply x^*, which is attained through solving (3). The solution map can then be formulated as $S(\Psi) = (R \circ s \circ C)(\Psi) = x^*$, where s maps the problem data (A, b, c) to a solution (x^*, ν^*) and $R(x^*, \nu^*) = x^*$. The gradient of the solution map is then formulated as $\nabla S(\Psi) = \nabla C(\Psi)\nabla s(A, b, c)\nabla R(x^*, \nu^*)$. We therefore only need to worry about $\nabla s(A, b, c)$ which, similar to [5], is attained through differentiating the KKT conditions of (3) and is derived in detail in [4] and [1]. The combination of cvxpy and differentiating through a cone program is then fully integrated in the

software package CvxpyLayers [2] which is used for all of the simulation results demonstrated in this report. Now we can perform an update on the problem data of our high level optimization problem, similar to (2), without the need for the problem to be in the standard form of (1). For example, if we wanted to build a differentiable MPC policy using CvxpyLayers, we set $\Psi = \{C_t, c_t, F_t, f_t\}_{1:T}$, $x = \tau_{1:T}$ and then define the update rule $C_t^{(k+1)} = C_t^{(k)} - \eta \nabla_{C_t} \mathcal{L}(S(\Psi))$, where $\nabla_{C_T} S(\Psi)$ is attained through CvxpyLayers.

3.3 Learning Convex Optimization Control Policies

Convex optimization control policies (COCPs) consider the class of control policies where optimal inputs are determined through solving a convex optimization problem. Examples include approximate dynamic programming in which the cost-to-go is constrained to be convex, and specific forms of MPC such as the linear quadratic regulator (LQR). Learning COCPs considers the subset of these problems where you are tasked with controlling a stochastic system with paramaterized dynamics and cost model [3]. Defining a cost on the parameters in the form of a utility function $\varphi : \Theta \to \mathbb{R}$, the task is to then find the parameters of the dynamics and/or cost function which minimizes the expected value of $\varphi(\theta)$ over a time horizon T, with $\theta \in \Theta$. In our case, we define φ as the closed loop cost of the system over the time horizon T. This leads to the formulation;

$$\theta^* = \arg\min_{\theta} \mathcal{L}(\theta), \;\; \mathcal{L}(\theta) = \mathbb{E}_\theta[\varphi(\theta)] \tag{4}$$

Of course, this is intractable to solve in closed form due to rarely having access to the distribution of the parameters over the closed loop cost. A monte-carlo estimate is then obtained in the form of batches, as described below.

$$\hat{\mathcal{L}}(\theta) = \frac{1}{B} \sum_{i=0}^{B-1} \varphi_i(\theta) \tag{5}$$

Where B is the batch size and $\varphi_i(\theta)$ refers to the closed loop cost of batch i. We describe each closed loop trajectory as a batch since we will be evaluating each of these closed-loop trajectories in parallel, utilizing the batch capabilities of CvxpyLayers. A projected gradient method is then utilized to iteratively tune θ.

$$\theta_{k+1} = \Pi_\Theta(\theta_k - n \nabla \hat{\mathcal{L}}(\theta_k)) \tag{6}$$

Where Θ defines the set of all feasible parameters and Π is the projection operator. Note that this method is not guaranteed to find the globally optimal parameters since this is a local search method and the cost is not a convex function of θ.

4 Preliminaries

4.1 Data Enabled Predictive Control

Data Enabled Predictive Control (DeePC) is a direct data driven control formulation which, when applied to deterministic LTI systems, has equivalent

guarantees to model predictive control (MPC) [7]. However, we naturally want to find a formulation that can adapt to both non-linear and stochastic systems. Fortunately, these require minimal effort to implement [8], [10]. The formulation that we will be working with for the remainder of this report is as follows:

$$\begin{aligned} \min_{y,u,g,\sigma_y,\sigma_u} \sum_{i=0}^{T-1} ||y_i - r_{t+i}||_Q^2 + ||u_i||_R^2 + \\ \theta_0 ||(I-\Pi)g||_2^2 + \theta_1|g|_1 + \theta_2|\sigma_y|_1 + \theta_3|\sigma_u|_1 \\ \text{subject to} \begin{pmatrix} U_p \\ Y_p \\ U_f \\ Y_f \end{pmatrix} g = \begin{pmatrix} u_{\text{ini}} \\ y_{\text{ini}} \\ u \\ y \end{pmatrix} + \begin{pmatrix} \sigma_u \\ \sigma_y \\ 0 \\ 0 \end{pmatrix} \; u \in \mathcal{U}, y \in \mathcal{Y} \end{aligned} \tag{7}$$

Where θ represents our problem parameters. The goal of the method presented in this paper is to find the parameters that minimize our closed-loop cost.

4.2 Differentiable Optimization

4.2.1 Backpropagation

Backpropagation (backprop) is an algorithm generally studied in the context of training neural networks [12]. The chain rule is recursively applied a loss function to evaluate the sensitivity of the loss with respect to the problem parameters. A first-order optimization algorithm is then applied to update the parameters at the next time step. Equation (8) illustrates how this can be applied to a generic loss function $\mathcal{L}$ which maps the solution of an implicit function g to a scalar loss value, using standard gradient descent with step size η. This example was chosen since this is how backpropagation is done in a convex optimisation layer where $g : X \times \Theta \to 0$ is the KKT conditions of (3).

$$\begin{aligned} \nabla_{\theta_k}\mathcal{L} &= -\nabla_{\theta_k} g(x,\theta_k)^\intercal \nabla_x g(x,\theta_k)^{-\intercal} \nabla_x \mathcal{L} \\ \theta_{k+1} &= \theta_k - \eta \nabla_{\theta_k}\mathcal{L} \end{aligned} \tag{8}$$

4.2.2 Resilient Backprop

Resilient backprop (Rprop)is a first order optimization algorithm that optimizes over each scalar element of the parameter vector individually using only the sign of the gradient, [11]. The update rule is as follows:

$$\begin{aligned} \theta_i^{(k+1)} &= \theta_i^{(k)} + \eta_i^{(k)} \text{sign}(\nabla \hat{\mathcal{L}}(\theta_i^{(k)}) \\ n_i^{(k)} &= \begin{cases} \min(\alpha \eta_i^{(k-1)}, \eta_{\max}) & \text{if } \nabla\mathcal{L}(\theta_i^{(k)})\nabla\mathcal{L}(\theta_i^{(k-1)}) < 0 \\ \max(\beta \eta_i^{(k-1)}, \eta_{\min}) & \text{if } \nabla\mathcal{L}(\theta_i^{(k)})\nabla\mathcal{L}(\theta_i^{(k-1)}) > 0 \\ \eta_i^{(k-1)} & \text{if } \nabla\mathcal{L}(\theta_i^{(k)})\nabla\mathcal{L}(\theta_i^{(k-1)}) = 0 \end{cases} \end{aligned} \tag{9}$$

Where $\eta_{\max} = 50$, $\eta_{\min} = 10^{-4}$, $\beta = 0.5$ and $\alpha = 1.2$. The intuition is that we increase/decrease our parameters exponentially, which is a necessary component of our method as it may be necessary to increase/decrease our parameters by orders of magnitude in order to stabilize the system.

5 Case Studies

Two systems were studied for verification of our method. The first being a marginally unstable LTI system with zero mean gaussian noise applied only to the outputs of the system. The second, a cartpole system with zero-mean gaussian noise applied to both the inputs and outputs. The LTI system was chosen as an initial proof of concept and the cartpole system for its non-linearity. When training both of the systems, we evaluate the performance based on the closed loop cost of the systems prediction horizon on a simple regulation task. Both systems are given a random initial condition for each batch which is within its state constraints and can be reached within its actuation limits given the time horizon. We demonstrate that this is enough for long term performance in the presence of disturbances.

5.1 Stochastic LTI System

This system represents a three room cooling system where the state of the system is the temperature of each room and the inputs represent a means of increasing the temperature in each room. The dynamics are described as folllows:

$$A = \begin{pmatrix} 1.01 & 0.01 & 0 \\ 0.01 & 1.01 & 0.01 \\ 0 & 0.01 & 1.01 \end{pmatrix} \; B, C = I_{3\times 3}$$
$$y_k = Cx_k + \omega_k, \; \omega_k \sim \mathcal{N}(0, 0.01)$$

We then formulate our DeePC problem as

$$\min_{y,u,g,\sigma_y} \sum_{i=0}^{T-1} ||y_i - r_{t+i}||_Q^2 + ||u_i||_R^2 + \theta|\sigma_y|_1$$
$$\text{subject to} \begin{pmatrix} U_p \\ Y_p \\ U_f \\ Y_f \end{pmatrix} g = \begin{pmatrix} u_{\text{ini}} \\ y_{\text{ini}} \\ u \\ y \end{pmatrix} + \begin{pmatrix} 0 \\ \sigma_y \\ 0 \\ 0 \end{pmatrix} \; u \in \mathcal{U}, y \in \mathcal{Y}$$

An estimate of our expected closed loop cost formulated as

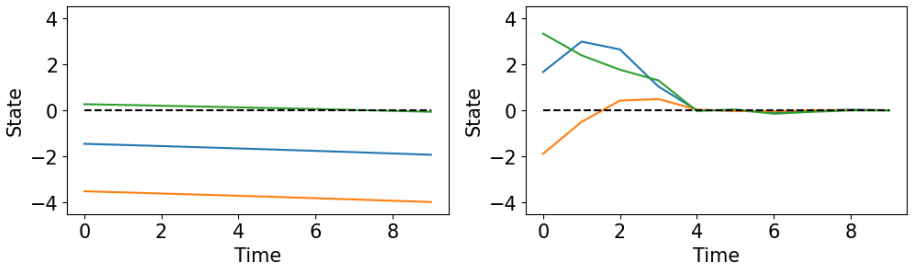
$$\hat{\mathcal{L}}(\theta) = \frac{1}{B} \sum_{j=0}^{B-1} \varphi_j(\theta), \quad \varphi_j(\theta) = \sum_{i=0}^{T-1} ||y_i^{*(j)}(\theta) - r_{t+i}||_Q^2 + ||u_i^{*(j)}(\theta)||_R^2 + |\sigma_y^{*(j)}(\theta)|_1$$

and parameter update, using resilient backprop,

$$\theta^{(k+1)} = \Pi_\Theta(\theta^{(k)} + \eta^{(k)} \text{sign}(\nabla \hat{\mathcal{L}}(\theta^{(k)})), \; \Theta = [10^{-3}, 10^3]$$

Where $\nabla \hat{\mathcal{L}}(\theta^{(k)}$ is attained through CvxpyLayers.

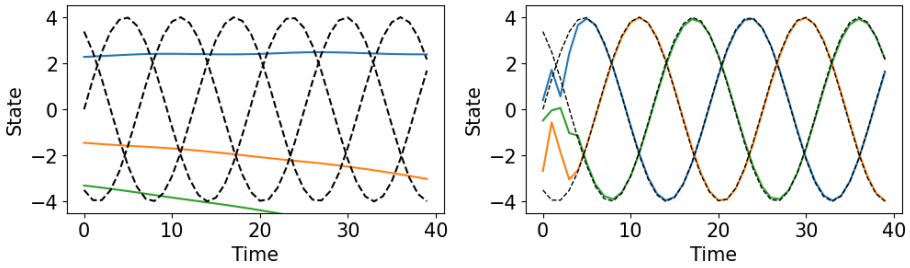
Figure 1 illustrates the systems regulation response at the beginning of training against after it converges at iteration 50. We set $\theta_0 = 0.001$ and at convergence we have $\theta_{50} = 32$. We also let $Q = 50I_{3\times3}$ and $R = 2I_{3\times3}$.



(a) Iteration 0 (b) Iteration 50

Figure 1: Regulation response of the system at either end of the simulation. Solid line represents trajectory and dashed line represents reference signal

After converging to the optimal parameters for its closed-loop regulation response we test the system on a time varying reference signal for 4 times the length of its prediction horizon to observe how it responds. As seen from Figure 2, the system demonstrates optimal behaviour.



(a) Iteration 0 (b) Iteration 50

Figure 2: System response to a time varying reference signal

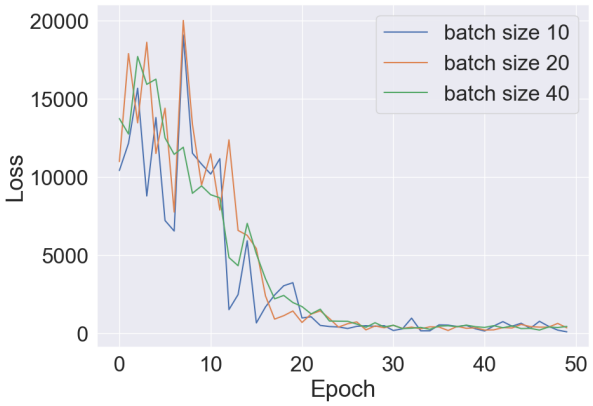


Figure 3: Training loss over time for stochastic LTI system

5.2 Stochastic Nonlinear System

Cartpole with state $s_k = (x_k, \dot{x}_k, \delta_k, \dot{\delta}_k)$, where $(x_k, \dot{x}_k)$ represent the horizontal displacement from the origin and linear velocity respectively at timestep k, $(\delta_k, \dot{\delta}_k)$ represent the angular displacement of the pole and angular velocity of the pole respectively and scalar input u_k corresponds to force applied to the cart in newtons.

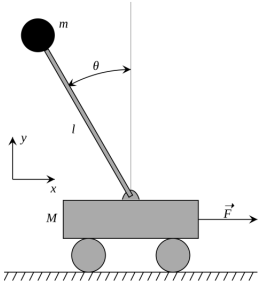


Figure 4: Figure of a cartpole system

Dynamics are then formulated as;

$$s_{k+1} = g(s_k, u_k + \zeta_k) + \omega_k$$
$$\zeta_k \sim \mathcal{N}(0, 10^{-4}),\ \omega_{k,i} \sim \mathcal{N}(0, 10^{-6})\ \forall\, i = 1, \ldots, 4$$

Since we are using all regularizatons, our DeePC problem is simply formulated as (3). However, due to the non-linearity of the system, we have to be careful when doing our offline data-collection. Since DeePC formulations assume that the data matrices are generated by a linear system, we have to ensure that they capture a linear approximation of the cartpole system. This is done by attempting to stabilize the system about the origin with a P controller and adding zero mean

Gaussian noise to the control inputs to ensure persistence of excitation. This was done in our experiments and the max absolute values of δ and x were ensured to be less than 0.15 radians and 0.2 meters. Once this is complete, the DeePC controller is formulated as stated in (7) and our optimization problem can then be formulated as;

$$\varphi_j(\theta) = \sum_{i=0}^{T-1} ||y_i^{*(j)}(\theta) - r_{t+i}||_Q^2 + ||u_i^{*(j)}(\theta)||_R^2 \\ + ||(I - \Pi)g^*(\theta)||_2^2 + |g^*(\theta)|_1 + |\sigma_y^*(\theta)|_1 + |\sigma_u^*(\theta)|_1$$

$$\theta^{(k+1)} = \Pi_\Theta(\theta^{(k)} + \eta^{(k)}\text{sign}(\nabla\hat{\mathcal{L}}(\theta^{(k)})), \quad \Theta = [10^{-3}, 10^3]^4$$

Figure 4 illustrates the convergence of the closed loop cost over time for the cartpole system where training consists of nothing but simple regulation tasks and $x_{\text{init}} = (0, 0, \epsilon, 0)$, $\epsilon \sim \text{uniform}(-0.01, 0.01)$. Parameters were initialised with a reasonable guess of $\theta_0 = (200, 200, 200, 200)$ with $Q = diag(100, 10, 100, 10)$ and $R = 0.01$. The parameters then converged to an optimal value of $\theta_{70} = (200.7, 0.87, 418.8, 200.1)$

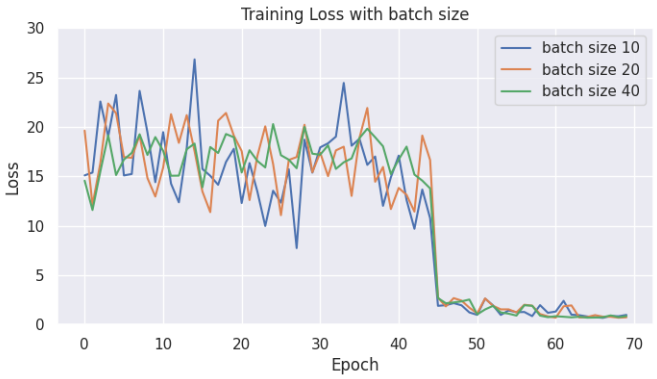


Figure 5: Training loss over time for stochastic cartpole system

The sudden drop in the loss function is a result of the method finding a θ that stabilizes the system. Since stability of the system mainly relies on θ_1 being small, iteration 45 represents the transition of θ_1 dropping from 50 to 0.001. In our experiments we observed that this method would fail to work when using any other gradient method that relied on the magnitude of the gradient.

Similarly to the stochastic LTI system, we would like to evaluate the controllers performance beyond a simple regulation task after the training procedure has converged. For the cartpole system, we injected a substantial disturbance at timestep 20 and let the system run for a further 130 time steps. The trajectory of the system (Figure 6) demonstrates the effectiveness of the controller beyond the simple regulation task presented in training. This behaviour wasn't surprising for either of the systems studied in this project as the controller already has

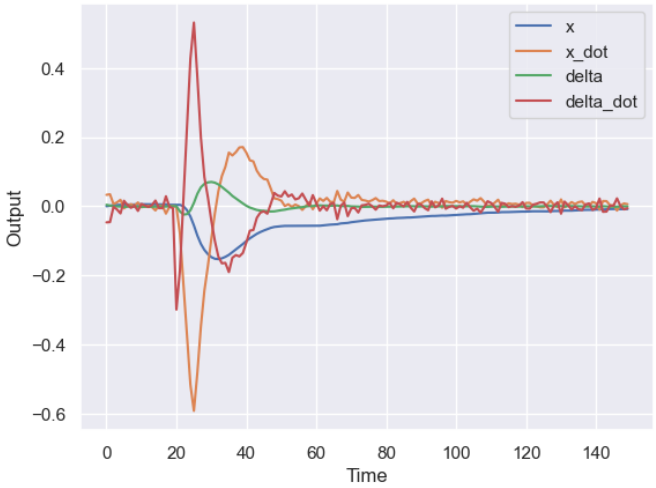


Figure 6: Trajectory of cartpole system with disturbance

a non-parametric approximation of the system dynamics in the form of data matrices.

6 Conclusions and Future Work

The method developed in this project shows the effectiveness of backpropagating through DeePC for hyperparameter tuning and that can be done in roughly 50 iterations. This can therefore be extended to an RL situation where the critic network can be replaced with a differentiable DeePC layer. Unfortunately the backward pass is quite slow since we have to take the inverse of a jacobian at each iteration. However, this is very new area of research in optimization theory and papers are being released regularly with different schemes to avoid this inversion [13].

References

[1] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers, 2019.

[2] A. Agrawal, B. Amos, S. Barratt, S. Boyd, S. Diamond, and Z. Kolter. Differentiable convex optimization layers. In *Advances in Neural Information Processing Systems*, 2019.

[3] A. Agrawal, S. Barratt, S. Boyd, and B. Stellato. Learning convex optimization control policies, 2019.

[4] A. Agrawal, S. Barratt, S. Boyd, E. Busseti, and W. M. Moursi. Differentiating through a cone program, 2020.

[5] B. Amos and J. Z. Kolter. Optnet: Differentiable optimization as a layer in neural networks, 2021.

[6] B. Amos, I. D. J. Rodriguez, J. Sacks, B. Boots, and J. Z. Kolter. Differentiable mpc for end-to-end planning and control, 2019.

[7] J. Coulson, J. Lygeros, and F. Dörfler. Data-enabled predictive control: In the shallows of the deepc, 2019.

[8] J. Coulson, J. Lygeros, and F. Dörfler. Regularized and distributionally robust data-enabled predictive control, 2019.

[9] S. Diamond and S. Boyd. CVXPY: A Python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[10] F. Dörfler, J. Coulson, and I. Markovsky. Bridging direct indirect data-driven control formulations via regularizations and relaxations, 2021.

[11] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. In *IEEE International Conference on Neural Networks*, pages 586–591 vol.1, 1993. doi: 10.1109/ICNN.1993.298623.

[12] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[13] H. Sun, Y. Shi, J. Wang, H. D. Tuan, H. V. Poor, and D. Tao. Alternating differentiation for optimization layers, 2023.