Understanding 'Optimizing Neural Networks via Koopman Operator Theory'

Abhinav Rao [*1]

Abstract

Optimization of neural network parameters can be visualized as a dynamical system in a discrete temporal domain. The Koopman operator is a powerful tool in the study of dynamical systems, with a recent resurgence in data-driven Koopman operator theory. It allows for linear dynamics in infinite dimensions that is equivalent to the non-linear dynamics in the finite state space. In this report we will explore the work of Dogra et al (1), which applies Koopman operator theory in neural network optimization. We will lay the theoretical groundwork, implement their method and discuss the validity of this method.

Introduction

Neural networks are universal approximators of continuous functions (2). They showcase many advantageous properties over other traditional approximators - They are relatively resistant against the curse of dimensionality and better computational efficiency due to their affine and batch-friendly structure. Robbins and Monro, proved that under certain basic constraints, a gradient descent algorithm will find the optimum weights that minimizes the approximation error(3). With these two assurances, neural networks and gradient-based optimization together form a basic universal approximation protocol for most well defined problems of the type $\mathbf{R}^m \rightarrow \mathbf{R}^n$. Yet, in practice, vanilla gradient descent faces several issues limiting the viability of neural networks. These include getting stuck in sub-optimal local minima and/or inflections, sensitivity to the choice of initial state, and poor robustness against improper learning rate selection.

The two major modifications to gradient descent that tackle these issues and are used ubiquitously in practice are the stochastic gradient descent (SGD) and momentum. The Koopman operator based approach proposed by (1), is an advanced extension to these methods that can circumvent a limited set of iterations using linear dynamics. To build understanding towards the application of Koopman operator for neural network optimization we follow these steps:

- Understand neural network optimization as a dynamical system.
- Describe the validity and advantages of modern optimization methods like SGD and momentum, through this lens of dynamical systems.
- Understand the basics of Koopman operators in the domain of dynamical systems.
- Explore an implementation of Koopman operator-driven neural network optimization on a non-trivial benchmark machine learning problem.

Links

Project Repo: Link.
Additional Repo: Link (Visualizing Momentum).
Original Paper Repo: Link (Matlab and partial codebase)

Background

In this section we explain the premise of optimization as a dynamical system problem and motivate the reader towards a Koopman approach.

1. Neural Network optimization as a Dynamical System:

The gradient descent algorithm is given as:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta f$$

Where, θ_n & θ_{n+1} are the states of the neural network weights during the nth and $n+1$th iteration. η is the learning rate and $\nabla_\theta f$ is the gradient of the loss function f. We can rearrange the terms as follow:

$$\frac{\theta_{n+1} - \theta_n}{\eta} = -\nabla_\theta f$$

Considering $\eta \rightarrow 0$ and $n \rightarrow \infty$, it is not difficult to see this equation is analogous to a dynamical system of the following form.

$$\lim_{\delta t \rightarrow 0} \frac{\theta_{t+\delta t} - \theta_t}{\delta t} = -\nabla_\theta f$$

$$\dot{\theta} = -\nabla_\theta f$$

We can make the following observations:

- Neural network optimization is analogous to a dynamical system with weights $\theta \in \Theta$ describing the state of the system.
- The loss function induces a potential energy field in the domain of weights. This is a scalar field, and the weights where the minimum of this field lies, are the desired optima.
- This field exerts a force with magnitude proportional to the magnitude of gradient of the field in the direction opposite to the gradient at any point.
- Following this gradient with appropriate dynamics 'may' allow the weights to reach the required optima autonomously.

2. Stochastic Gradient Descent and Momentum:

Stochastic gradient descent and mini-batch gradient descent are effective in getting out of inflections and shallow minimums and are also more robust against the initial state choice. They achieve this by sampling the data points on which loss is calculated each iteration. This can be visualized in the dynamical system view, by imagining a stochastic field of potential energy, since the loss term changes a bit each iteration. The loss field is a random field because the data originates from the same distribution. Consider the general case of a mini-batch gradient descent of the form:

$$\theta_{n+1} = \theta_n - \eta \nabla_\theta f(X_{i:i+m})$$

Where, i is essentially dependent on n, since the data changes each iteration. This is equivalent to a stochastic dynamical system of the form:

$$\dot{\theta} = -\nabla_\theta f(X(t))$$

Momentum on the other hand improves the optimization efficacy by changing the dynamics altogether. It achieves this by making the acceleration proportional to the force exerted by the field, unlike gradient descent which relies on velocity being proportional to the force. This modification results in 'inertia', i.e. the velocity does not become zero the moment the force (gradient) becomes zero. This inertia allows the optimizer to escape inflections and shallow minimas. Consider the following dynamical system, where the acceleration is proportional to the net force which includes the force exerted by the potential energy field and a drag force against the movement:

$$\ddot{\theta} = -\nabla f - \gamma \dot{\theta}$$

This can be written as a ODE in the standard form using a new variable for velocity

$$\dot{\theta} = v$$

$$\dot{v} = -\nabla f - \gamma v$$

This can be shown to be equivalent to the equations of gradient descent with momentum:

$$\theta_{n+1} = \theta_n + v_{n+1}$$

$$v_{n+1} = \beta v_n - \alpha \nabla_\theta f$$

3. The Koopman Operator Theory: The Koopman operator theory involves defining observables, which are functions of the state, and then identifying the dynamics of these observables in their infinite dimensional function space. Under some constraints, these observables follow linear dynamics which can be described a linear operator colloquially called the Koopman operator. In this section we follow the description of the original paper and describe the Koopman operator treatment for a discrete time dynamical system.

Consider a m dimensional state vector $\mathbf{x}$, such that some discrete dynamical system $(\mathcal{M}, t, T)$ be parameterized by the state space variables $\mathbf{x} = [x_1, x_2, .., x_m \ \in \mathcal{M} \subset \mathbf{R}^m$, where the evolution of $\mathbf{x}$ over discrete time $t \in Z$ is governed by the dynamical map $T : \mathcal{M} \to \mathcal{M}$. With this we can say:

$$\mathbf{x}(t+1) = T(\mathbf{x}(t))$$

The authors assume the systems under study are autonomous (T is not explicitly dependent on t). Now let $g : \mathcal{M} \to \mathbf{C}$ be the observable of interest. Here, we assume that $g \in \mathcal{F} = \mathcal{L}^2(\mathcal{M}, \sigma)$, where σ is a positive single value function that acts as the measure for the functional space. The Koopman operator is defined to be the object $K : \mathcal{F} \to \mathcal{F}$ that supplies the following action for g,

$$Kg(\mathbf{x}) = g \cdot T(\mathbf{x})$$

K is an object that lifts the dynamics from the original, possibly nonlinear, dynamical system $(\mathcal{M}, t, T)$ to a new, certainly linear, dynamical system $(\mathcal{F}, t, K)$. This can be then extended to multiple observables $\mathbf{g} = [g_1, g_2, ..., g_k]$.

$$K\mathbf{g}(\mathbf{x}) = \mathbf{g} \cdot T(\mathbf{x})$$

As we know, linear dynamics are a solved domain in dynamical systems, but it is important to note that K is an infinite dimensional operator.

Method

In the previous section we laid out the theoretical premise of visualizing neural network optimization and introduced the Koopman operator theory. In (1), the authors tie these together in an implementation by making three choices:

1. The choice of observables: The authors choose the identity function $\mathbf{I}$ as choice of observable for the Koopman treatment. This allows directly evaluating the dynamics of the weights, since the dynamics of the observables are the dynamics of the state. We note here, that the authors actually use the projection function $\pi_i(\mathbf{x}) = x_i$ as the observable, and use all the projection functions $(\pi_1, \pi_2, ...\pi_m)$ to build the observable list, since the identity observable $\mathbf{I}(\mathbf{w}) = \mathbf{w}$, as defined in the paper, that returns the state vector does not satisfy the $\mathcal{L}^2$ requirement of the observable.

2. Choice of Koopman operator approximation method: To approximate the Koopman operator, the authors use the finite section method, which uses two shifted matrix of all collected observables. That is, $F = [\mathbf{g}_1, \mathbf{g}_2, \mathbf{g}_3, ...\mathbf{g}_{n-1}]$ and $F' = [\mathbf{g}_2, \mathbf{g}_2, \mathbf{g}_3, ...\mathbf{g}_n]$, then the Koopman operator K can be approximated by $\hat{K}$:

$$\hat{K} = F'F^+$$

Where F^+ is Moore Penrose pseudo-inverse of F, given by $F^+ = (F^\dagger F)^{-1}F^\dagger$.

3. Computational compromise: The theoretically coherent thing to do would be to record every weight of the neural network as a column of the matrix, and subsequently apply the finite section method on them. The authors posit the computational complexity of this might be be intractable due to the size of the neural network and thus provide a spectrum of sub-optimal approximation. This spectrum spans building the matrix (a scalar) for each weight, for each node, for each layer or for the entire network (the right thing to do). The authors use node-wise dynamics justifying it with the trade-off against computational complexity. Here a node, refers to all the weights from a layer i used for a single activation in layer $i + 1$.

Results

I implemented the MNIST (4) simulation of the original paper. The code for this is **not** available, but the paper contains most of parameters required for re-implementation. The original paper uses quasi-node method to predict the dynamics of the first layer. Instead we use full-node method. Although, this is a deviation, it is a more generous and theoretically coherent method. The following steps list the algorithm for the implementation:

- Train the neural network for t_1 epochs using a conventional optimizer like AdaDelta (5).
- After that, continue this till t_2^{th} epoch but while recording all the parameters.
- After t_2 epochs, use the recorded parameters to build the Koopman matrix(s) using the appropriate method (Finite Section).
- For the next T epochs use Koopman operator update to update the parameters. (Which is just a matrix vector product).

Instead of Matlab we used JAX with equinox. This allows us to just-in-time compile the neural network part of the code and make decent time-savings. The results for the full-node and full-layer KOT simulations on the MNIST data sets for $t_1 = 3$, $t_2 = 5$, $T = 1$ and $T = 2$ is given in figure 1, figure 2, figure 3, and figure 4.

Discussion

The results from our python simulations do not match the MATLAB simulations of the original paper. Unlike the original paper, our implementation indicates a deviation in the losses as iterations increase. This can be attributed to exponentially growing or decaying weights. This is expected since the dynamical system of the observables is linear, but the states are same as observables, and thus they too follow exponential growth/decay. The authors display results in which the weights follow a non-linear path, which indicates deviation in our implementation and the actual implementation of the Koopman matrix which is not reported in detail. Additionally, using JIT the time differences skew in the favour of conventional optimizers. But this could be attributed to poor code optimization on our part. Overall, the performance is comparable for 1 epoch, but anything above that we can see the performance exponentially deteriorates (figure 3 and figure 4). On the theoretical side, there is the question of using identity observables as the authors state, or the projection observables as we state. Unless the observables span a Koopman invariant subspace, the approximation is not guaranteed, but in this case the choice might be too shallow to capture the dynamics of the loss landscape.

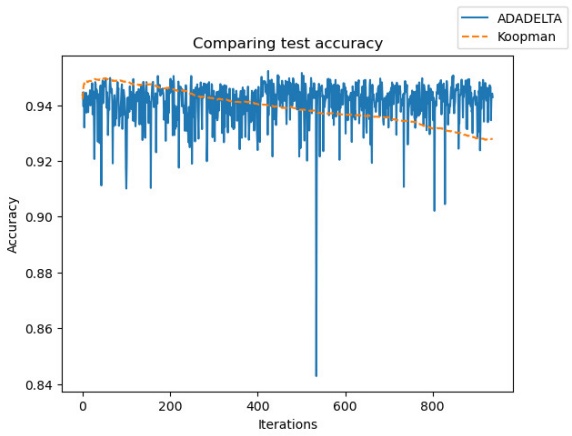
References

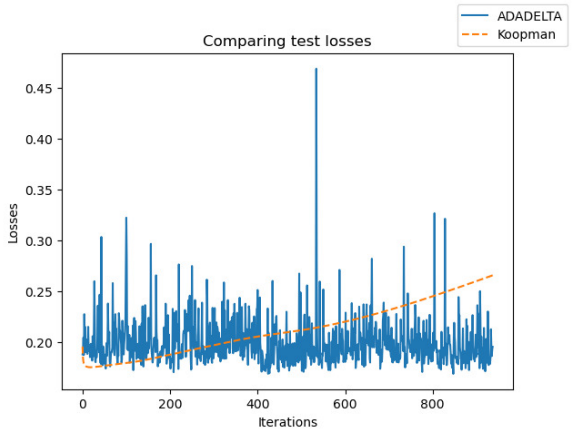
[1] Akshunna S Dogra and William Redman. Optimizing neural networks via koopman operator theory. *Advances in Neural Information Processing Systems*, 33:2087–2097, 2020.

[2] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[3] Herbert Robbins and Sutton Monro. A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407, 1951.

[4] Li Deng. The mnist database of handwritten digit images for machine learning research [best of the





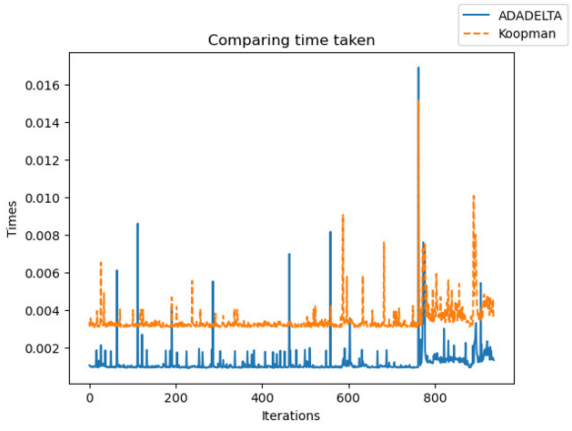
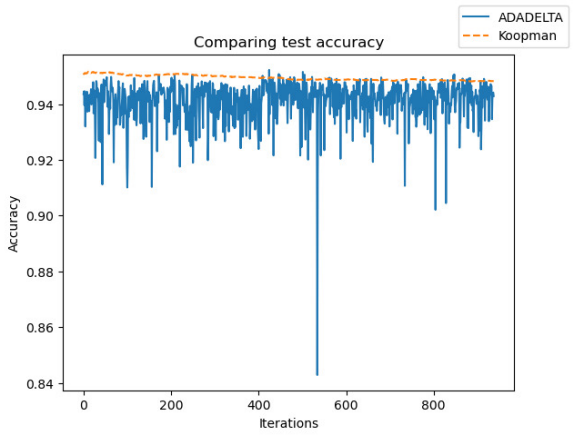
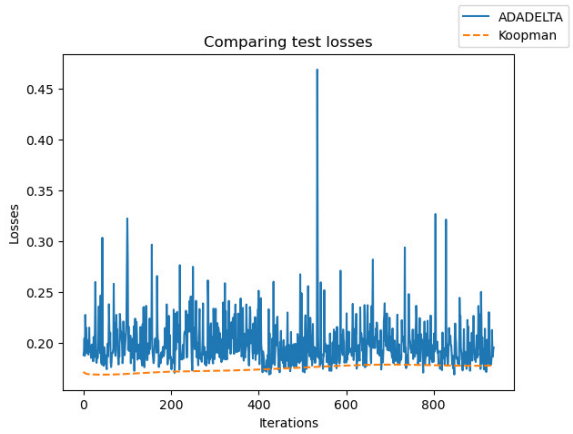


Figure 1. Comparison of performance of the Node-wise Koopman operator optimization against AdaDelta for 1 epoch (a) Test accuracy (b) Test losses (c) Training times





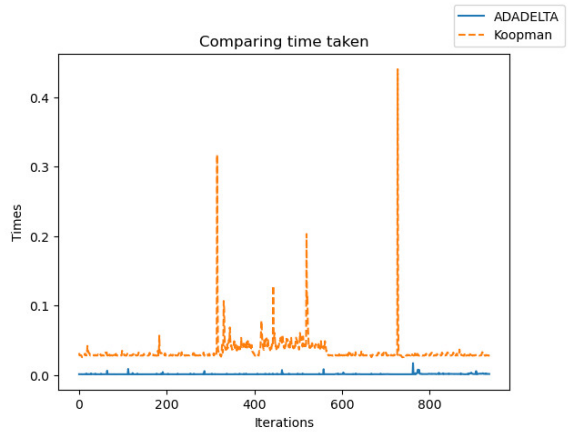
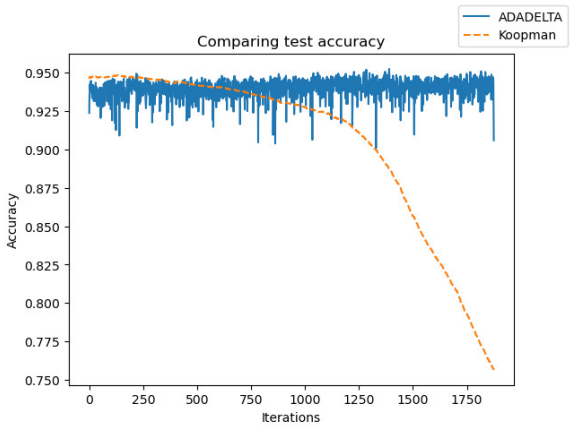
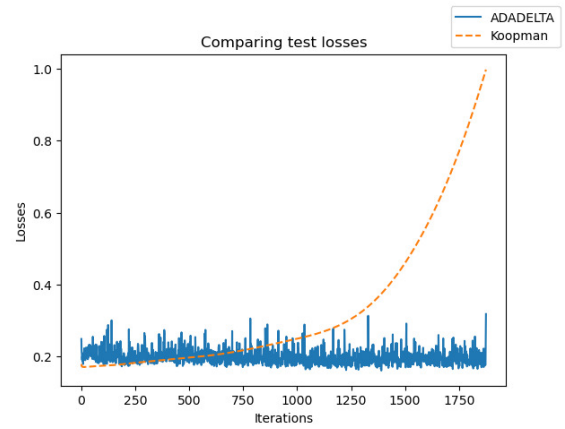


Figure 2. Comparison of performance of the Layer-wise Koopman operator optimization against AdaDelta for 1 epoch (a) Test accuracy (b) Test losses (c) Training times





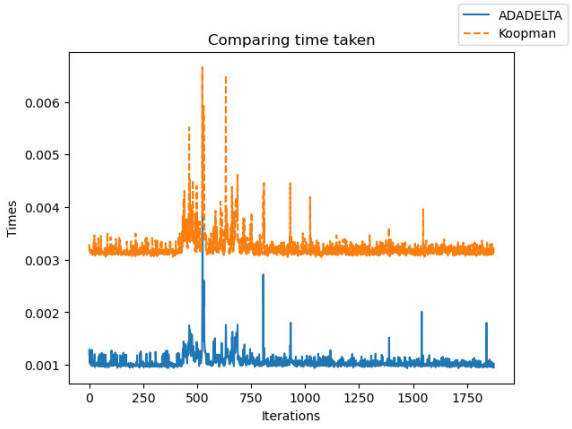
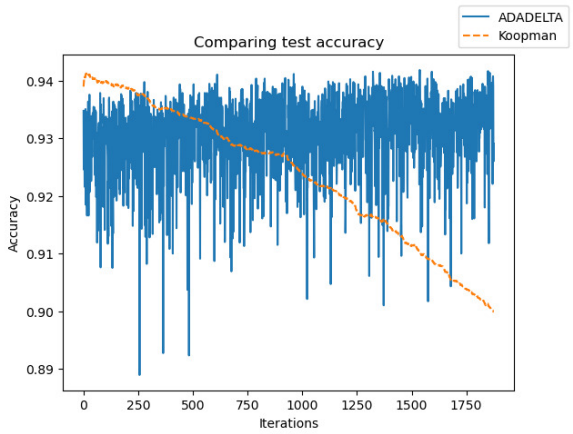
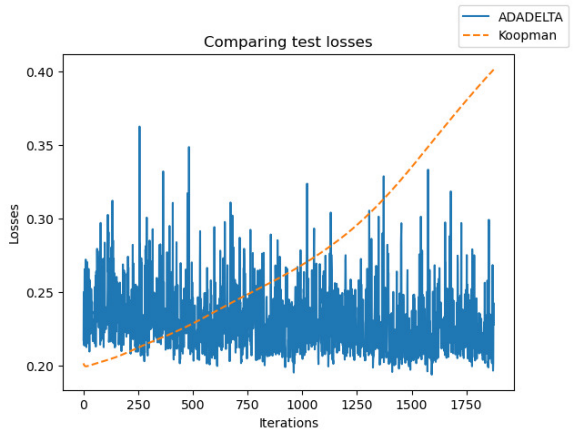


Figure 3. Comparison of performance of the Node-wise Koopman operator optimization against AdaDelta for 2 epochs (a) Test accuracy (b) Test losses (c) Training times





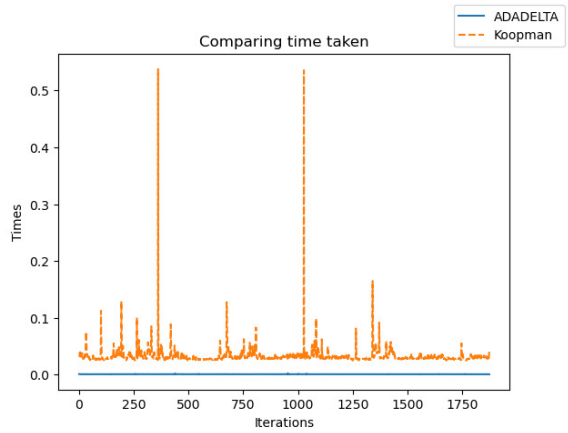


Figure 4. Comparison of performance of the Layer-wise Koopman operator optimization against AdaDelta for 2 epochs (a) Test accuracy (b) Test losses (c) Training times

web]. *IEEE signal processing magazine*, 29(6):141–142, 2012.

[5] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.