Deep Declarative Networks: Further Explorations

Mokai Pan
ShanghaiTech University
2021533106
panmk@shanghaitech.edu.cn

Huaqiu Wang
ShanghaiTech University
2021533143
wanghq@shanghaitech.edu.cn

Shouchen Zhou
ShanghaiTech University
2021533042
zhoushch@shanghaitech.edu.cn

Abstract

We have explored the deep declarative networks (DDNs), a novel class of deep learning models that define network layers in terms of desired behavior through mathematical optimization, rather than explicit function definitions. By leveraging the implicit function theorem, DDNs enable gradient back-propagation through declaratively defined nodes, facilitating end-to-end learning. This approach allows the integration of classical optimization techniques within a learnable framework, extending the functionality of neural network layers to include optimization-based operations. The versatility and effectiveness of DDNs are demonstrated through theoretical insights and applications in image and point cloud classification, as well as some situations set and solving bi-level optimization problems of hyperparameter optimization explored by our own.

1 Introduction

Recent years have witnessed a proliferation of machine learning and artificial intelligence techniques [1], [2], [3], among which deep neural networks (DNNs) have been particularly prominent. The effectiveness of these methods often hinges on fitting highly overparameterized models that contain more parameters than the training data points, enabling them to achieve near-zero training errors on specific tasks.

However, the explicit definition of network layers limits their adaptability and integration with optimization-based methods. Deep declarative networks (DDNs) [4] propose a paradigm shift by using optimization problems to implicitly define the behavior of network layers. This paper introduces DDNs, discussing their theoretical foundations, implementation in PyTorch, and application potential. By integrating optimization directly into the learning process, DDNs offer a flexible and powerful framework for building advanced neural architectures that can solve complex, real-world problems in an end-to-end trainable manner.

2 Related Work

In our own explorations, we tried to reproduce the test of the comparisons between DNN's running time and memory consumption and the normal networks. [5] had also done such case studies: robust vector pooling and optimal transport, which inspired us.

Based on DDN, [6] explores under what circumstances the gradient of deep declaration nodes can be approximated by ignoring constraint terms.

SI251 Convex Optimization Final Project.

[7] as well as [4] has also tried to use DDN to solve bi-level problems. These were used as the base code for our exploration of argument selection problems with regression with hyperparameter regularization, which is a bi-level optimization problem. We used these base codes to construct our own methods and calculated the best regularization hyperparameters.

3 Methods

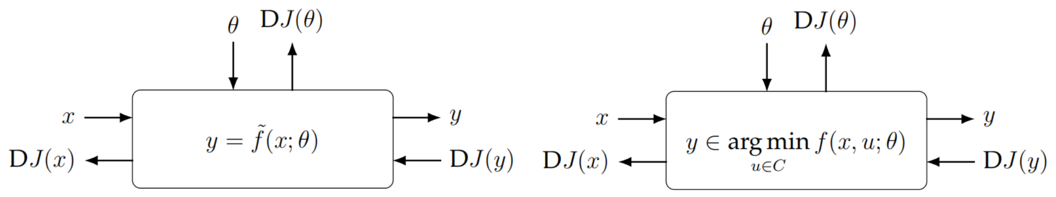

Figure 1: Schematic diagram of Deep Declarative Network

A declarative node is one in which the exact implementation of the forward processing function is not defined; rather the input-output relationship $(x \mapsto y)$ is defined in terms of behavior specified as the solution to an optimization problem

$$y \in \arg\min_{u \in C} f(x, u; \theta)$$

where f is the (parametrized) objective function, θ are the node parameters (possibly empty), and C is the set of feasible solutions, i.e. the constraints.

For the constrained of the optimization problem, it could be classified into Unconstrained(3.1.1), Equality Constrained(3.1.2), Inequality Constrained(3.1.3) three situations. The details are shown below below in 3.1.

3.1 Back-propagation through declarative nodes

Consider a function f in the objective :

$$f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$$

And the equality function h (if equality constraints exist):

$$h : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^p$$

And the inequality function g (if inequality constraints exist):

$$g : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^q$$

Then we can separately discuss the situations in whether these constraints exist or not.

3.1.1 Unconstrained

Let

$$y(x) \in \arg\min_{u \in \mathbb{R}^m} f(x, u).$$

Assume $y(x)$ exists and that f is second-order differentiable in the neighborhood of the point $(x, y(x))$. Set $H = \mathrm{D}^2_{YY} f(x, y(x)) \in \mathbb{R}^{m \times m}$ and $B = \mathrm{D}^2_{XY} f(x, y(x)) \in \mathbb{R}^{m \times n}$. Then for H non-singular the derivative of y with respect to x is

$$\mathrm{D}_y(x) = -H^{-1}B.$$

3.1.2 Equality Constrained

Let

$$\begin{aligned} y(x) \in \arg\min_{u \in \mathbb{R}^m} \;& f(x, u) \\ \text{subject to} \;\; & h_i(x, u) = 0, \; i = 1, \dots, p. \end{aligned}$$

Assume that $y(x)$ exists, that f and $h = [h_1, \ldots, h_p]^\top$ are second-order differentiable in the neighborhood of $(x, y(x))$, and that $\text{rank}(\mathrm{D}_Y h(x, y)) = p$. Then for H non-singular

$$\mathrm{D}_y(x) = H^{-1}A^\top \left(AH^{-1}A^\top\right)^{-1}(AH^{-1}B - C) - H^{-1}B$$

where

$$\begin{aligned}
A &= \mathrm{D}_Y h(x, y) \in \mathbb{R}^{p\times m} \\
B &= \mathrm{D}^2_{XY} f(x, y) - \sum_{i=1}^{p} \lambda_i \mathrm{D}^2_{XY} h_i(x, y) \in \mathbb{R}^{m\times n} \\
C &= \mathrm{D}_X h(x, y) \in \mathbb{R}^{p\times n} \\
H &= \mathrm{D}^2_{YY} f(x, y) - \sum_{i=1}^{p} \lambda_i \mathrm{D}^2_{YY} h_i(x, y) \in \mathbb{R}^{m\times m}
\end{aligned}$$

and $\lambda \in \mathbb{R}^p$ satisfies $\lambda^\top A = \mathrm{D}_Y f(x, y)$.

3.1.3 Inequality Constrained

Let

$$\begin{array}{ll}
y(x) \in \arg\min\limits_{u\in\mathbb{R}^m} & f(x, u) \\
\text{subject to} & h_i(x, u) = 0,\ i = 1, \ldots, p \\
 & g_i(x, u) \leq 0,\ i = 1, \ldots, q.
\end{array}$$

Assume that $y(x)$ exists, that f, h and g are secondorder differentiable in the neighborhood of $(x, y(x))$,and that all inequality constraints are active at $y(x)$. Let $\tilde{h} = [h_1, \ldots, h_p, \hat{g_1}, \ldots, g_q]$ and assume $\text{rank}(\mathcal{D}_Y \tilde{h}(x, y)) = p + q$. Then for $\hat{H}$ non-singular

$$\mathrm{D}_y(x) = H^{-1}A^{\mathrm{T}} \left(AH^{-1}A^{\mathrm{T}}\right)^{-1} \left(AH^{-1}B - C\right) - H^{-1}B$$

where

$$\begin{aligned}
A &= \mathrm{D}_Y \tilde{h}(x, y) \in \mathbb{R}^{(p+q)\times m} \\
B &= \mathrm{D}^2_{XY} f(x, y) - \sum_{i=1}^{p+q} \lambda_i \mathrm{D}^2_{XY} \tilde{h}_i(x, y) \in \mathbb{R}^{m\times n} \\
C &= \mathrm{D}_X \tilde{h}(x, y) \in \mathbb{R}^{(p+q)\times n} \\
H &= \mathrm{D}^2_{YY} f(x, y) - \sum_{i=1}^{p+q} \lambda_i \mathrm{D}^2_{YY} \tilde{h}_i(x, y) \in \mathbb{R}^{m\times m}
\end{aligned}$$

and $\lambda \in \mathbb{R}^{p+q}$ satisfies $\lambda^{\mathsf{T}} A = D_Y f(x, y)$ with $\lambda_i \leq 0$ for $i = p+1, \ldots, p+q$. The gradient $D_y(x)$ is one-sided whenever there exists an $i > p$ such that $\lambda_i = 0$.

3.2 Illustrative Examples

3.2.1 Robust Pooling

Average (or mean) pooling is a standard operation in deep learning models where multi-dimensional feature maps are averaged over one or more dimensions to produce a summary statistic of the input data. For the one-dimensional case, $\mathbf{x} \in \mathbb{R}^n \to y \in \mathbb{R}$, we have $y = \frac{1}{n}\sum_{i=1}^{n} x_i$ and $D_y = \left[\frac{\partial y}{\partial x_1}, \frac{\partial y}{\partial x_2}, ..., \frac{\partial y}{\partial x_n}\right] = \frac{1}{n}\mathbf{1}^{\mathrm{T}}$.

We can write it as a declarative node:

$$y(x) \in \arg\min_{u\in\mathbb{R}} \sum_{i=1}^{n} \frac{1}{2}(u - x_i)^2$$

While the solution, and hence gradients, can be expressed in closed form, it is well-known that as a summary statistic, the mean is very sensitive to outliers. We can make the statistic more robust by replacing the quadratic penalty function $\phi^{\text{quad}}(z) = \frac{1}{2}z^2$ with one that is less sensitive to outliers. We can generalize the pooling operation as

$$y(x) \in \arg\min_{u\in\mathbb{R}} \sum_{i=1}^{n} \frac{1}{2}\phi(u - x_i; \alpha)$$

For example, the Welsch penalty function

$$\phi^{\text{Welsch}}(z; \alpha) = 1 - e^{-\frac{z^2}{2\alpha^2}}$$

3.2.2 Sphere Projection

Euclidean projection onto an L2-sphere, equivalent to L2 normalization, is another standard operation in deep learning models. For $\mathbf{x} \in \mathbb{R}^n \to y \in \mathbb{R}^n$, we have $y = \frac{1}{\|x\|_2}x$ and $D_y = \frac{1}{\|x\|_2}\left(\mathrm{I} - \frac{1}{\|x\|_2}xx^{\mathrm{T}}\right)$

Also, we can write it as a declarative node:

$$\begin{aligned} y \in \arg\min_{u\in\mathbb{R}^n} &\frac{1}{2}\|u - x\|_2^2 \\ \text{s.t. } &\|u\|_2 = 1 \end{aligned} \tag{1}$$

We can also generalize to other L_p-spheres to improve generalization performance as:

$$\begin{aligned} y \in \arg\min_{u\in\mathbb{R}^n} &\frac{1}{2}\|u - x\|_2^2 \\ \text{s.t. } &\|u\|_p = 1 \end{aligned} \tag{2}$$

where $\|\cdot\|_p$ is the L_p-norm.

For example, projecting onto the L_1-sphere:

$$\begin{aligned} y \in \arg\min_{u\in\mathbb{R}^n} &\frac{1}{2}\|u - x\|_2^2 \\ \text{s.t. } &\sum_{i=1}^{n}|u_i| = 1 \end{aligned} \tag{3}$$

and projecting onto the L_∞-sphere:

$$\begin{aligned} y \in \arg\min_{u\in\mathbb{R}^n} &\frac{1}{2}\|u - x\|_2^2 \\ \text{s.t. } &\max_{1\le i\le n}|u_i| = 1 \end{aligned} \tag{4}$$

When used to define a projection operation, only the L_2 case has a closed-form solution. Nonetheless, the gradient of the solution concerning the input data for all constraint functions can be calculated using 3.1.2 Equality Constrained.

3.3 Bi-level optimization problems of hyperparameter optimization

Many machine learning models have hyperparameters, for example, the Ridge Regression model and the Support Vector Machine model. We mainly focus on the Ridge Regression model:

$$\min_{\beta} \frac{1}{2}\|X\beta - y\|_2^2 + \frac{1}{2}\mu\|\beta\|_2^2 \tag{5}$$

where $X \in \mathbb{R}^{n\times d}$ is a matrix composed of n training data with a d-dimension feature, $y \in \mathbb{R}^n$ is the related label, μ is the hyperparameter and $\beta \in \mathbb{R}^d$ is the parameter vector of the model.

In the Ridge Regression, how to choose a better hyperparameter, also known as hyperparameter optimization, is a critical issue. We take a common method for hyperparameter optimization, which is K-fold cross-validation. [8]

K-fold cross-validation is done by dividing the training dataset $D := \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ into K parts $\mathbf{D}_k := \{(\mathbf{x}_i, y_i)\}_{i\in\mathbf{T}_k}$, where $\mathbf{T}_k$, $k = 1, 2, ..., \mathrm{K}$ is a division of the index set $\mathbf{T} := \{1, 2, ..., n\}$. For a fixed μ and each k, we use $\mathbf{D}_k^{'} := \mathbf{D}\backslash\mathbf{D}_k$ as the training dataset to solve the Ridge Regression model mentioned above 5 and we denote the resulting solution as $\beta_k(\mu)$. Denote X_k and $X_k^{'}$ as the data in $\mathbf{D}_k$ and $\mathbf{D}_k^{'}$ respectively and denote y_k and $y_k^{'}$ as the related label. Thus for each $k = 1, \cdots, \mathrm{K}$ after doing the same steps, we can compute the cross-validation error:

$$CV(\mu) = \frac{1}{\mathrm{K}}\sum_{k=1}^{\mathrm{K}}\frac{1}{2}\|X_k\beta_k(\mu) - y_k\|_2^2 \tag{6}$$

A classical approach for hyperparameter optimization with cross-validation is the grid search method, where one needs to define a grid over the hyperparameters of interest and search for the combination of hyperparameters that minimizes the cross-validation error. However, one of the drawbacks of the grid search approach is that the continuity of the hyperparameter is ignored by the discretization.[9] A formulation of the bilevel optimization model is proposed to choose hyperparameters.[10] To improve it, we come up with the idea of using bi-level optimization to solve it. Below, we will focus on the bi-level optimization of the hyperparameter optimization of the Ridge Regression model combined with DDN.

We can first write the hyperparameter optimization as a bi-level optimization problem:

$$\begin{aligned} &\min_{\mu,\beta_1,\beta_2,...,\beta_{\mathrm{K}}} \frac{1}{\mathrm{K}}\sum_{k=1}^{\mathrm{K}}\frac{1}{2}\|X_k\beta_k(\mu) - y_k\|_2^2 \\ &\text{s.t. } (\beta_1, \beta_2, ..., \beta_{\mathrm{K}}) \in \underset{(z_1,z_2,...,z_k)}{\arg\min} \sum_{k=1}^{\mathrm{K}}\frac{1}{2}\|X_k^{'}z_k - y_k^{'}\|_2^2 + \frac{1}{2}\mu\|z_k\|_2^2 \end{aligned} \tag{7}$$

The lower-level problem can be solved by DDN. If we denote $\beta = (\beta_1, \beta_2, ..., \beta_{\mathrm{K}}) \in \mathbb{R}^{Kd}$, which is a long vector composed of every β_k, objective function $f(\beta, \mu) = \sum_{k=1}^{\mathrm{K}}\frac{1}{2}\|X_k^{'}\beta_k - y_k^{'}\|_2^2 + \frac{1}{2}\mu\|\beta_k\|_2^2$ and loss function $J(\beta(\mu)) = \frac{1}{\mathrm{K}}\sum_{k=1}^{\mathrm{K}}\frac{1}{2}\|X_k\beta_k(\mu) - y_k\|_2^2$, according to method 3.1.1 (the unconstrained situation) in Deep Declarative Network, we can learn that the solution of lower-level problem is

$$\frac{\partial f}{\partial \beta} = 0 \Rightarrow \beta_k(\mu) = (X_k^{'\mathrm{T}}X_k^{'} + \mu I)^{-1}\cdot(X_k^{'\mathrm{T}}y_k^{'}) \tag{8}$$

and the derivative of $\beta_k(\mu)$ with respect to μ

$$\begin{aligned} \frac{\partial \beta_k(\mu)}{\partial \mu} &= -(\frac{\partial^2 f}{\partial \beta_k^2})^{-1}\cdot\frac{\partial^2 f}{\partial\beta_k\partial\mu} \\ &= -(X_k^{\prime\top}X_k^{\prime} + \mu I)^{-1}\beta_k \end{aligned} \tag{9}$$

and also we can get the derivative of the loss function J concerning μ

$$\frac{\partial J(\beta(\mu))}{\partial \mu} = \sum_{k=1}^{\mathrm{K}}\frac{\partial J}{\partial \beta_k}\cdot\frac{\partial \beta_k}{\partial \mu} \tag{10}$$

Then, to solve the upper-level optimization problem, we take the GD(Gradient Descent) method to optimize the hyperparameter μ. In each iteration time t, the update equation is

$$\mu^{t+1} = \mu^t + (-1) \cdot \eta \cdot \frac{\partial J}{\partial \mu^t} \tag{11}$$

where $\eta > 0$ is the step size.

4 Experiments

We did experiments on the ModelNet40 and ImageNet(Mini) datasets.

4.1 ModelNet40

We reproduced the experiment of different robust types in the paper with a setting outline to be 0.6.

The results are shown in Figure 2, with the training loss, testing accuracy, and testing mAP illustrated. For the robust pooling types: we use 'Q' for Quadratic, 'PH' for Pseudo-Huber, 'H' for Huber, 'W' for Welsch, 'TQ' for Truncated Quadratic, and 'None' for the default settings, i.e. max-pooling.

The result implies that applying all these types of robust pooling improves the performance metrics: training loss, testing accuracy, and testing mAP than the default settings, consistent with the experiment conducted in the paper.

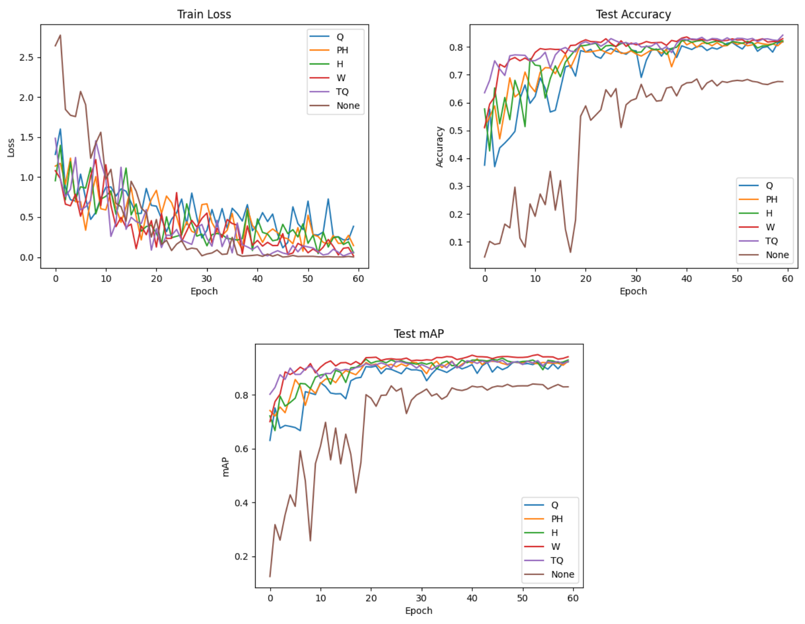

Figure 2: Experiments for different penalty functions: Point Cloud classification on ModelNet40 with penalty functions: Q, PH, H, W, TQ, None. A detailed explanation of these functions is mentioned in the section 4.1. The training loss, testing accuracy, and testing mAP are illustrated.

4.2 ImageNet

We reproduced the experiment of L_p sphere projection conducted in the paper. Due to limited computing resources, we use the Mini-ImageNet dataset instead of the ImageNet used in the paper. Mini-ImageNet is a subset of ImageNet with 60,000 images in total, evenly distributed in 100 classes. All the models in the experiment are trained from scratch.

The results are shown in Figure 3, with mean average precision(mAP), top-1 accuracy, and top-5 accuracy on the validation set as metrics.For the models with L_p projection, we use scaling factors

$r = 250, 15, 3$ respectively for L_1, L_2 and L_∞ projections. The result implies that feature projection improves the performance evaluated by mAP, which is consistent with the experiment conducted in the paper. In the inaugural experiments, the top-1 accuracy and top-5 accuracy of the residual networks augmented with L_p sphere projection demonstrated a modest enhancement. Subsequently, our reproduction of the experiment revealed no discernible disparity in the top-1 accuracy and top-5 accuracy metrics when comparing the various experimental groups. This observed phenomenon may be attributed to the propensity of residual networks to exhibit a heightened susceptibility to overfit with diminutive datasets.

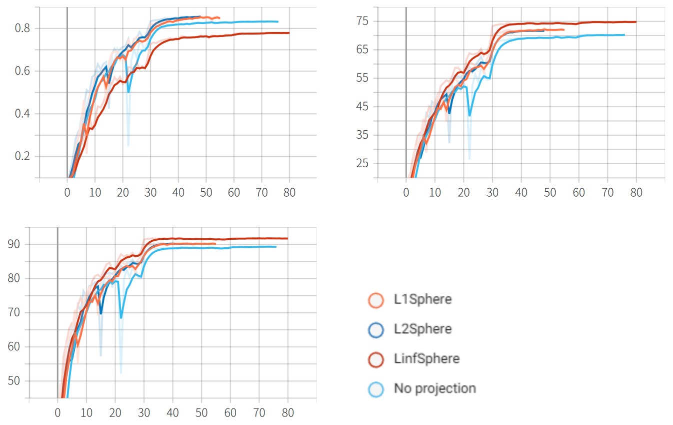

Figure 3: Experiments for L_p sphere projection: Image classification on Mini-ImageNet, ResNet18 with L_1, L_2, L_∞ spehere projection . The mAP, top-1 accuracy and top-5 accuracy are illustrated.

4.3 Bi-level problem of hyperparameter optimization

We conduct experiments on the iris dataset and the diabetes dataset loaded from scikit-learn, a commonly used Python module. The two datasets are commonly used datasets for classification and regression experiments, to assess the viability of applying declarative networks on hyperparameter optimization of Ridge Regression. The algorithm is shown as follows:

Gradient descent for hyperparameter optimization

```
from ddn.basic.node import *
from sklearn import datasets

def GradientDescent(node,mu_init, y_test_splitted , X_test_splitted , step_size=1.0e-1, tol=1.0
        e-8,max_iters=1500000, verbose=False):
    mu = mu_init
    history = []
    for i in range(max_iters):
        # solve the lower-level problem and compute the derivative of upper-level loss
          function to mu

        beta,_ = node.solve(mu)
        history.append( function_objective (beta, X_test_splitted , y_test_splitted ))
        if verbose: print("{:5d}: {}".format(i,history[-1]))
        if (len( history ) > 2) and ( history[-2] - history[-1]) < tol:
            break
        dJdx = 0
        for j in range(K):
            dJdx += np.dot((  derivative_objective_beta (beta, X_test_splitted ,
          y_test_splitted )[j]).T,node.gradient(miu, beta)[j])

        # take a step in the negative gradient direction
        mu -= step_size*dJdx
    return mu, history
```

We trained the two datasets with two initial values of the hyperparameter μ: $\mu_0 = 0.1$ and $\mu_0 = 800$, step size $\eta = 1.0 \times 10^{-1}$.

As for the iris dataset, after about one million iterations, we finally got an optimized hyperparameter $\mu \approx 345$, and its loss function value is about 0.7194. We visualize the gradient descent process as Figure 4:

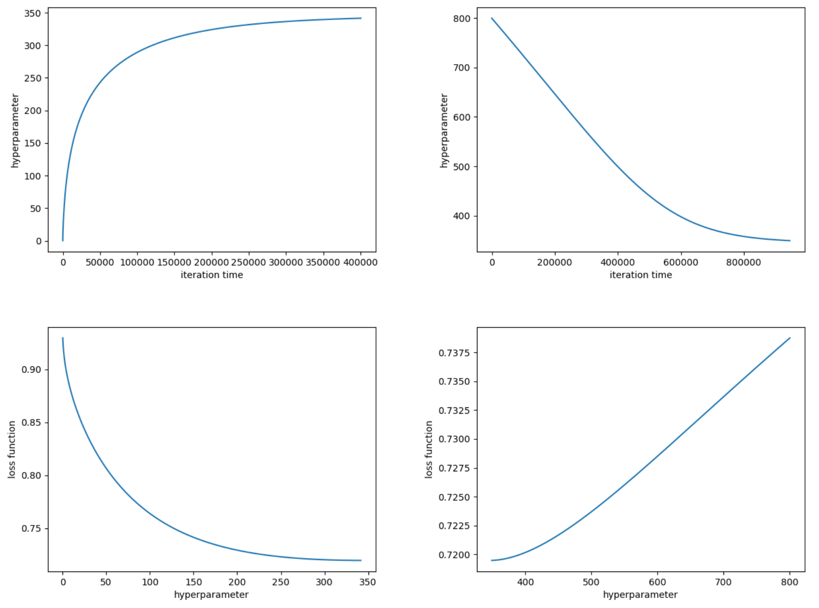

Figure 4: The training curves of iris dataset

As for the diabetes dataset, after several thousand iterations, we finally got an optimized hyperparameter $\mu \approx 210.147$, and its loss function value is about 0.901. We visualize the gradient descent process as Figure 5:

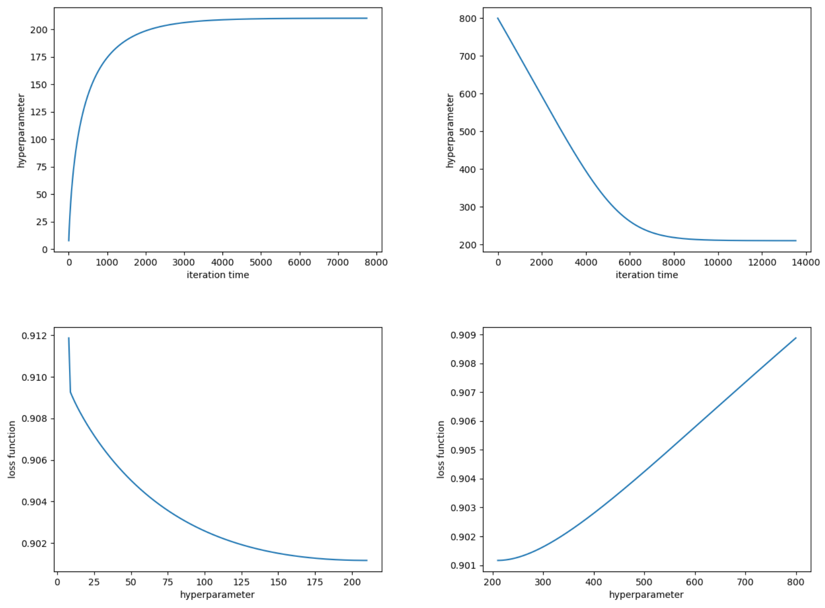

Figure 5: The training curves of diabetes dataset

4.4 Consumption

The consumption of running time and memory usage of the forward and backward process among different types of penalty functions were shown in figure 6. Time (top) and memory (bottom) requirements for forward and backward passes of robust vector pooling on the CPU. The 2D feature map ($\sqrt{n} \times \sqrt{n}$) has $m = 128$ channels, and the batch size is one. We use L-BFGS in the forward pass except for quadratic, which has a closed-form solution. For non-convex penalties, we take the best solution from two different initializations. The backward pass is implemented by implicit differentiation.

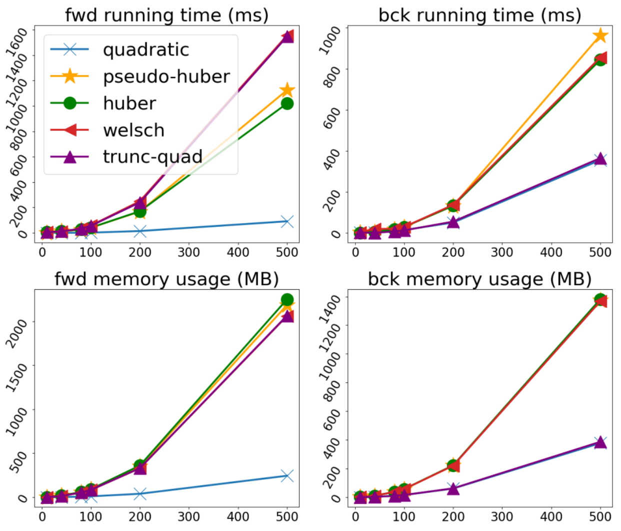

Figure 6: The consumption of running time, the memory usage of the forward and backward process when training on ModelNet40.

5 Conclusion

Deep Declarative Networks explores a new class of end-to-end learnable models where the forward function is implicitly defined as the solution to a mathematical optimization problem. The class of deep declarative networks subsumes current deep learning models.

In the preceding sections, according to the several main methods3.13.2 in the Deep Declarative Network, we have reproduced some theories and practices for DDNs and conducted experiments of different robust types with ModelNet40 CAD dataset4.1 and L_p sphere projection with Mini-ImageNet dataset4.2 and targeted to the similar result from the paper.

Then, we attempted to use DDN to solve bi-level problems of hyperparameter optimization of the Ridge Regression3.3 and conduct experiments on the iris dataset and the diabetes dataset. The two results of the optimized hyperparameter are really good.

In the process of experiments, we also explore the relationship between the consumption of the running time and memory usage of the forward and backward process4.4.

Also, there are still some shortcomings and much still to do. For example, it is possible for problems with parametrized constraints to become infeasible during learning, and we have made no assumption about the uniqueness of the solution to the optimization problem, and if the Hessian matrix is singular, then we may not get an isolated minimizer.

Finally, our future works may focus on researching whether the performance would be better and the consumption would be less when we change the nodes in other deep learning networks into deep declarative nodes and explore new methods and algorithms that could get the same performance with a lower price. We may also consider testing our hyperparameter optimization based on bi-level and ddn on bigger dataset and tuning other hyperparameters including step size to improve our algorithm.

References

[1] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia Chen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. 2019.

[2] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby. Big transfer (bit): General visual representation learning. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part V 16*, pages 491–507. Springer, 2020.

[3] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning (still) requires rethinking generalization. *Communications of the ACM*, 64(3):107–115, 2021.

[4] Stephen Gould, Richard Hartley, and Dylan Campbell. Deep declarative networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44:3988–4004, Aug 2022.

[5] Stephen Gould, Dylan Campbell, Itzik Ben-Shabat, Chamin Hewa Koneputugodage, and Zhiwei Xu. Exploiting problem structure in deep declarative networks: Two case studies, 2022.

[6] Stephen Gould, Ming Xu, Zhiwei Xu, and Yanbin Liu. Towards understanding gradient approximation in equality constrained deep declarative networks, 2023.

[7] Stephen Gould, Basura Fernando, Anoop Cherian, Peter Anderson, Rodrigo Santa Cruz, and Edison Guo. On differentiating parameterized argmin and argmax problems with application to bi-level optimization, 2016.

[8] Kristin P. Bennett, Gautam Kunapuli, Jing Hu, and J. S. Pang. Bilevel optimization and machine learning. In *IEEE World Congress on Computational Intelligence*, 2008.

[9] Qingna Li, Zhen Li, and Alain Zemkoho. Bilevel hyperparameter optimization for support vector classification: theoretical analysis and a solution method, 2021.

[10] G. Kunapuli, K. P. Bennett, Jing Hu, and Jong-Shi Pang. Classification model selection via bilevel programming. 2008.