

Optimization of Graph Neural Networks with Natural Gradient Descent

Mohammad Rasool Izadi ^{*†} Yihao Fang [†] Robert Stevenson ^{*} Lizhen Lin [†]
mizadi@nd.edu yfang5@nd.edu rls@nd.edu lizhen.lin@nd.edu

^{*}Electrical Engineering, [†]Applied and Computational Mathematics and Statistics
University of Notre Dame
Notre Dame, IN, USA

Abstract

In this work, we propose to employ information-geometric tools to optimize a graph neural network architecture such as the graph convolutional networks. More specifically, we develop optimization algorithms for the graph-based semi-supervised learning by employing the natural gradient information in the optimization process. This allows us to efficiently exploit the geometry of the underlying statistical model or parameter space for optimization and inference. To the best of our knowledge, this is the first work that has utilized the natural gradient for the optimization of graph neural networks that can be extended to other semi-supervised problems. Efficient computations algorithms are developed and extensive numerical studies are conducted to demonstrate the superior performance of our algorithms over existing algorithms such as ADAM and SGD.

Index Terms

Graph neural network, Fisher information, natural gradient descent, network data.

I. INTRODUCTION

In machine learning, the cost function is mostly evaluated using labeled samples that are not easy to collect. Semi-supervised learning tries to find a better model by using unlabeled samples. Most of the semi-supervised methods are based on a graph representation on (transformed) samples and labels [11]. For example, augmentation methods create a new graph in which original and augmented samples are connected. Graphs, as datasets with linked samples, have been the center of attention in semi-supervised learning. Graph Neural Network (GNN), initially

proposed to capture graph representations in neural networks [20], have been used for semi-supervised learning in a variety of problems like node classification, link predictions, and so on. The goal of each GNN layer is to transform features while considering the graph structure by aggregating information from connected or neighboring nodes. When there is only one graph, the goal of node classification becomes predicting node labels in a graph while only a portion of node labels are available (even though the model might have access to the features of all nodes). Inspired by the advance of convolutional neural networks [14] in computer vision [12], Graph Convolutional Network (GCN) [10] employs the spectra of graph Laplacian for filtering signals and the kernel can be approximated using Chebyshev polynomials or functions [24, 22]. GCN has become a standard and popular tool in the emerging field of geometric deep learning [2].

From the optimization perspective, Stochastic Gradient Descent (SGD)-based methods that use an estimation of gradients have been popular choices due to their simplicity and efficiency. However, SGD-based algorithms may be slow in convergence and hard to tune on large datasets. Adding extra information about gradients, may help with the convergence but are not always possible or easy to obtain. For example, using second-order gradients like the Hessian matrix, resulting in the Newton method, is among the best choices which, however, are not easy to calculate especially in NNs. When the dataset is large or samples are redundant, NNs are trained using methods built on top of SGD like AdaGrad [4] or Adam [9]. Such methods use the gradients information from previous iterations or simply add more parameters like momentum to the SGD. Natural Gradient Descent (NGD) [1] provides an alternative based on the second-moment of gradients. Using an estimation of the inverse of the *Fisher information matrix* (simply Fisher), NGD transforms gradients into so-called *natural gradients* that showed to be much faster compared to the SGD in many cases. The use of NGD allows efficient exploration of the geometry of the underlying parameter space in the optimization process. Also, Fisher information plays a pivotal role throughout statistical modeling [16]. In frequentist statistics, Fisher information is used to construct hypothesis tests and confidence intervals by maximum likelihood estimators. In Bayesian statistics, it defines the Jeffreys’s prior, a default prior commonly used for estimation problems and nuisance parameters in a Bayesian hypothesis test. In minimum description length, Fisher information measures the model complexity and its role in model selection within the minimum description length framework like AIC and BIC. Under this interpretation, NGD is invariant to any smooth and invertible reparameterization of the model, while SGD-based

methods highly depend on the parameterization. For models with a large number of parameters like DNN, Fisher is so huge that makes it almost impossible to evaluate natural gradients. Thus, for faster calculation it is preferred to use an approximation of Fisher like Kronecker-Factored Approximate Curvature (KFAC) [18] that are easier to store and inverse.

Both GNN and training NNs with NGD have been active areas of research in recent years but, to the best of our knowledge, this is the first attempt on using NGD in the semi-supervised learning. In this work, a new framework for optimizing GNNs is proposed that takes into account the unlabeled samples in the approximation of Fisher. Section II provides an overview of related topics such as semi-supervised learning, GNN, and NGD. The proposed algorithm is described in section III and a series of experiments are performed in section IV to evaluate the method's efficiency and sensitivity to hyper-parameters. Finally, the work is concluded in section V.

II. PROBLEM AND BACKGROUND

In this section, first, the graph-based semi-supervised learning with a focus on least-squared regression and cross-entropy classification is defined. Required backgrounds on the optimization and neural networks are provided in the subsequent sections. A detailed description of the notation is summarized in the Table I.

A. Problem

Consider an information source $q(\mathbf{x})$ generating independent samples $\mathbf{x}_i \in \mathbb{X}$, the target distribution $q(\mathbf{y}|\mathbf{x})$ associating $\mathbf{y}_i \in \mathbb{Y}$ to each $\mathbf{x}_i$, and the adjacency distribution $q(a|\mathbf{x}, \mathbf{x}')$ representing the link between two nodes given their covariates levels $\mathbf{x}$ and $\mathbf{x}'$. The problem of learning $q(\mathbf{y}|\mathbf{x})$ is to estimate some parameters $\boldsymbol{\theta}$ that minimizes the cost function

$$r(\boldsymbol{\theta}) = E_{\mathbf{x}, \mathbf{x}' \sim q(\mathbf{x}), \underline{a} \sim q(a|\mathbf{x}, \mathbf{x}'), \mathbf{y} \sim q(\mathbf{y}|\mathbf{x})} [l(\mathbf{y}, \mathbf{f}(\mathbf{x}, \underline{\mathbf{x}}', \underline{a}; \boldsymbol{\theta}))] \quad (1)$$

where the loss function $l(\mathbf{y}, \hat{\mathbf{y}})$ measures the prediction error between $\mathbf{y}$ and $\hat{\mathbf{y}}$. Also, $\underline{\mathbf{x}}'$ and $\underline{a}$ show sequences of $\mathbf{x}'$ and a , respectively. As $q(\mathbf{x})$, $q(a|\mathbf{x}, \mathbf{x}')$, and $q(\mathbf{y}|\mathbf{x})$ are usually unknown or unavailable, the cost $r(\boldsymbol{\theta})$ is estimated using samples from these distributions. Furthermore, it is often more expensive to sample from $q(\mathbf{y}|\mathbf{x})$ than $q(\mathbf{x})$ and $q(a|\mathbf{x}, \mathbf{x}')$ resulting in the different number of samples from each distribution being available.

Let $X = X_0$ to be a $d_0 \times n$ matrix of $n \geq 1$ i.i.d $\mathbf{x}_i$ samples from $q(\mathbf{x})$ (equivalent to $X \sim q(X)$). It is assumed that $n \times n$ adjacency matrix $A = [a_{ij}]$ is sampled from $q(a|\mathbf{x}_i, \mathbf{x}_j)$ for $i, j = 1, \dots, n$ (equivalent to $A \sim q(A|X)$). One can consider (X, A) to be a graph of n

TABLE I
NOTATION

Symbol	Description
$x, \mathbf{x}, X$	Scalar, vector, matrix
$\epsilon, \lambda, \gamma$	Regularization hyper-parameters
η	The learning rate
A	Adjacency matrix
$X^\top$	Matrix transpose
I	Comfortable identity matrix
$\underline{\mathbf{x}}$	A sequence of $\mathbf{x}$ vectors
n	The total number of samples
$\bar{n}$	The number of labeled samples
F	Fisher information matrix
B	Preconditioning matrix
$r(\boldsymbol{\theta})$	The cost of parameters $\boldsymbol{\theta}$
$l(\mathbf{y}, \hat{\mathbf{y}})$	The loss between $\mathbf{y}$ and $\hat{\mathbf{y}}$
$q(\mathbf{x})$	The source distribution
$q(\mathbf{y} \mathbf{x})$	The target distribution
$q(a \mathbf{x}, \mathbf{x}')$	The adjacency distribution
$p(\mathbf{y} f(X, A; \boldsymbol{\theta}))$	The prediction distribution
$\phi(\cdot)$	An element-wise nonlinear function
$\nabla_{\boldsymbol{\theta}} f$	Gradient of scalar f wrt. $\boldsymbol{\theta}$
$J_{\boldsymbol{\theta}} \mathbf{f}$	Jacobian of vector $\mathbf{f}$ wrt. $\boldsymbol{\theta}$
$H_{\boldsymbol{\theta}} f$	Hessian of scalar f wrt. $\boldsymbol{\theta}$
$\odot$	Element-wise multiplication operation

nodes in which the i th column of X shows the covariate at the node i and $D = \text{diag}(\sum_j a_{ij})$ denotes the diagonal degree matrix. Also, denote Y to be a $d_m \times \bar{n}$ matrix of $\bar{n} < n$ samples $\mathbf{y}_i$ from $q(\mathbf{y}|\mathbf{x}_i)$ for $i = 1, \dots, \bar{n}$ and $\mathbf{z} = [\mathbb{1}(i \in \{1, \dots, \bar{n}\})]_{i=1}^n$ to be the training mask vector. Note that $\mathbb{1}(\text{condition})$ is 1 if the condition is true and 0 otherwise. Thus, an empirical cost can be estimated by

$$\hat{r}(\boldsymbol{\theta}) = \frac{1}{\bar{n}} \sum_{i=1}^{\bar{n}} l(\mathbf{y}_i, \mathbf{f}(\mathbf{x}_i, X, A; \boldsymbol{\theta})), \quad (2)$$

where $\mathbf{f}(\mathbf{x}_i, X, A; \boldsymbol{\theta})$ shows the processed $\mathbf{x}_i$ when having access to $n - 1$ extra samples and links between them. Note that as X contains $\mathbf{x}_i$ (the i th column), $\mathbf{f}(\mathbf{x}_i, X, A; \boldsymbol{\theta})$ and $\mathbf{f}(X, A; \boldsymbol{\theta})$ are used interchangeably.

Assuming $p(\mathbf{y}|\mathbf{f}(X, A; \boldsymbol{\theta})) = p_{\boldsymbol{\theta}}(\mathbf{y}|X, A)$ to be an exponential family with natural parameters in $\mathbb{F}$, the loss function becomes

$$l(\mathbf{y}, \mathbf{f}(X, A; \boldsymbol{\theta})) = -\log p(\mathbf{y}|\mathbf{f}(X, A; \boldsymbol{\theta})). \quad (3)$$

In the least-squared regression,

$$p(\mathbf{y}|\mathbf{f}(X, A; \boldsymbol{\theta})) = \mathcal{N}(\mathbf{y}|\mathbf{f}(X, A; \boldsymbol{\theta}), \sigma^2) \quad (4)$$

for fixed σ^2 and $\mathbb{F} = \mathbb{Y} = \mathbb{R}$. In the cross-entropy classification to c classes,

$$p(y = k|\mathbf{f}(X, A; \boldsymbol{\theta})) = \exp(\mathbf{f}_k) / \sum_{j=1}^c \exp(\mathbf{f}_j) \quad (5)$$

for $\mathbb{F} = \mathbb{R}^c$ and $\mathbb{Y} = \{1, \dots, c\}$.

B. Parameter estimation

Having the first order approximation of $r(\boldsymbol{\theta})$ around a point $\boldsymbol{\theta}_0$,

$$r(\boldsymbol{\theta}) \approx r(\boldsymbol{\theta}_0) + \mathbf{g}_0^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_0), \quad (6)$$

the gradient descent can be used to update parameter $\boldsymbol{\theta}$ iteratively:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta B \mathbf{g}_0 \quad (7)$$

where $\eta > 0$ denotes the learning rate, $\mathbf{g}_0 = \mathbf{g}(\boldsymbol{\theta}_0)$ is the gradient at $\boldsymbol{\theta}_0$ for

$$\mathbf{g}(\boldsymbol{\theta}) = \frac{\partial r(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} \quad (8)$$

and B shows a symmetric positive definite matrix called *preconditioner* capturing the interplay between the elements of $\boldsymbol{\theta}$. In SGD, $B = I$ and $\mathbf{g}_0$ is approximated by:

$$\hat{\mathbf{g}}_0 = \frac{1}{\bar{n}} \sum_{i=1}^{\bar{n}} \frac{\partial l(y_i, \mathbf{f}(X, A; \boldsymbol{\theta}))}{\partial \boldsymbol{\theta}} \quad (9)$$

where $\bar{n} \geq 1$ can be the mini-batch (a randomly drawn subset of the dataset) size.

To take into the account the relation between $\boldsymbol{\theta}$ elements, one can use the second order approximation of $r(\boldsymbol{\theta})$:

$$r(\boldsymbol{\theta}) \approx r(\boldsymbol{\theta}_0) + \mathbf{g}_0^\top (\boldsymbol{\theta} - \boldsymbol{\theta}_0) + \frac{1}{2} (\boldsymbol{\theta} - \boldsymbol{\theta}_0)^\top H_0 (\boldsymbol{\theta} - \boldsymbol{\theta}_0), \quad (10)$$

where $H_0 = H(\boldsymbol{\theta}_0)$ denotes the Hessian matrix at $\boldsymbol{\theta}_0$ for

$$H(\boldsymbol{\theta}) = \frac{\partial^2 r(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}^\top \partial \boldsymbol{\theta}}. \quad (11)$$

Thus, having the gradients of $r(\boldsymbol{\theta})$ around $\boldsymbol{\theta}$ as:

$$\mathbf{g}(\boldsymbol{\theta}) \approx \mathbf{g}_0 + H_0(\boldsymbol{\theta} - \boldsymbol{\theta}_0), \quad (12)$$

the parameters can be updated using:

$$\boldsymbol{\theta}_{t+1} = (I - \eta B H_0) \boldsymbol{\theta}_t - \eta B(\mathbf{g}_0 - H_0 \boldsymbol{\theta}_0). \quad (13)$$

The convergence of Eq. 13 heavily depends on the selection of η and the distribution of $I - \eta B H_0$ eigenvalues. Note that update rules Eqs. 7 and 13 coincides at $B = H_0^{-1}$ resulting the Newton's method. As it is not always possible or desirable to obtain Hessian, several preconditioners are suggested to adapt the information geometry of the parameter space.

In NGD, the preconditioner is defined to be the inverse of Fisher Information matrix:

$$F(\boldsymbol{\theta}) := E_{\mathbf{x}, \mathbf{y} \sim p(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^T] \quad (14)$$

$$= E_{\mathbf{x} \sim q(\mathbf{x}), \mathbf{y} \sim p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})} [\nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^T] \quad (15)$$

where $p(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}) := q(\mathbf{x})p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})$ and

$$\nabla_{\boldsymbol{\theta}} := -\nabla_{\boldsymbol{\theta}} \log p(\mathbf{x}, \mathbf{y}; \boldsymbol{\theta}). \quad (16)$$

C. Neural Networks

A neural network is a mapping from the input space $\mathbb{X}$ to the output space $\mathbb{F}$ through a series of m layers. Layer $k \in \{1, \dots, m\}$, projects d_{k-1} -dimensional input $\mathbf{x}_{k-1}$ to d_k -dimensional output $\mathbf{x}_k$ and can be expressed as:

$$\mathbf{x}_k = \phi_k(W_k \mathbf{x}_{k-1}) \quad (17)$$

where ϕ_k is an element-wise non-linear function and W_k is the $d_k \times d_{k-1}$ -dimensional weight matrix. The bias is not explicitly mentioned as it could be the last column of W_k when $\mathbf{x}_k$ has an extra unit element. Let the $\boldsymbol{\theta} = [\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_m]$ to be the parameters of an m -layer neural network formed by stacking m vectors of dimension $d_k d_{k-1}$ for $k = 1, \dots, m$ and $\dim(\mathbf{x}) = d_0$ such that $\dim(\boldsymbol{\theta}) = \sum_{k=1}^m d_k d_{k-1}$. The parameters of the k 'th layer, $\boldsymbol{\theta}_k = \text{vec}(W_k)$ for $\text{vec}(W_k) = [\mathbf{w}_1, \dots, \mathbf{w}_{d_k}]$, is also shaped by piling up rows of W_k . Their gradients, $\nabla_{\boldsymbol{\theta}_k}$, could be written as:

$$\nabla_{\boldsymbol{\theta}_k} = \frac{\partial l}{\partial \boldsymbol{\theta}_k} = \frac{\partial \mathbf{x}_k}{\partial \boldsymbol{\theta}_k}^T \frac{\partial l}{\partial \mathbf{x}_k} \quad (18)$$

for $d_k \times d_k d_{k-1}$ -dimensional matrix $\partial \mathbf{x}_k / \partial \boldsymbol{\theta}_k$ and d_k -dimensional vector $\partial l / \partial \mathbf{x}_k$.

D. Graph Neural Networks

The Graph Neural Network (GNN) extends the NN mapping to the data represented in graph domains [20]. The basic idea is to use related samples when the adjacency information is available. In other words, the input to the k 'th layer, $\mathbf{x}_{k-1}$ is transformed into $\tilde{\mathbf{x}}_{k-1}$ that take into the account unlabeled samples using the adjacency such that $p(\mathbf{x}_{k-1}, A) = p(\tilde{\mathbf{x}}_{k-1})$. Therefore, for each node $i = 1, \dots, n$, the Eq. 17 can be written by a local transition function (or a single message passing step) as:

$$\mathbf{x}_{k,i} = \mathbf{f}_k(\mathbf{x}_{k-1,i}, \underline{\mathbf{x}}_{k-1,i}, \mathbf{x}_0, \underline{\mathbf{x}}_0; W_k) \quad (19)$$

where $\underline{\mathbf{x}}_{k,i}$ denotes all the information coming from nodes connected to the i th node at the k th layer. The subscripts here are used to indicate both the layer and the node, i.e. $\mathbf{x}_{k,i}$ means the state embedding of node i in the layer k . Also, the local transition Eq. 19, parameterized by W_k , is shared by all nodes that includes the information of the graph structure, and $\mathbf{x}_0 = \mathbf{x}_i$.

The Graph Convolutional Network (GCN) is a one of the GNN with the message passing operation as a linear approximation to spectral graph convolution, followed by a non-linear activation function as:

$$\mathbf{x}_{k,i} = \mathbf{f}_k(\mathbf{x}_{k-1,i}, \underline{\mathbf{x}}_{k-1,i}; W_k) \quad (20)$$

$$X_k = \phi_k(W_k X_{k-1} \tilde{A}) \quad (21)$$

$$= \phi_k(W_k \tilde{X}_{k-1}) \quad (22)$$

where ϕ_k is a element-wise nonlinear activation function such as $\text{RELU}(x) = \max(x, 0)$, W_k is a $d_k \times d_{k-1}$ parameter matrix that needs to be estimated. $\tilde{A}$ denotes the normalized adjacency matrix defined by:

$$\tilde{A} = (D + I)^{-1/2}(A + I)(D + I)^{-1/2} \quad (23)$$

to overcome the overfitting issue due to the small number of labeled samples $\bar{n}$. In fact, a GCN layer is basically a NN (Eq. 17) where the input $\mathbf{x}_{k-1}$ is initially updated into $\tilde{\mathbf{x}}_{k-1}$ using a so-called renormalization trick such that $\tilde{\mathbf{x}}_{k-1,i} = \sum_{j=1}^n \tilde{a}_{i,j} \mathbf{x}_{k-1,j}$ where $\tilde{A} = [\tilde{a}_{i,j}]$. Comparing Eq. 20 with the more general Eq. 19, the local transition function $\mathbf{f}_k$ is defined as a linear combination followed by a nonlinear activation function. For classifying $\mathbf{x}$ into c classes, having

a c -dimensional $\mathbf{x}_m$ as the output of the last layer with a Softmax activation function, the loss between the label $\mathbf{y}$ and the prediction $\mathbf{x}_m$ becomes:

$$l(\mathbf{y}, \mathbf{x}_m) = - \sum_{j=1}^c \mathbb{1}(\mathbf{x}_{m,j} = j) \log \mathbf{x}_{m,j}. \quad (24)$$

III. METHOD

The basic idea of preconditioning is to capture the relation between the gradients of parameters ∇_{θ} . This relation can be as complete as a matrix B (for example, NGD) representing the pairwise relation between elements of ∇_{θ} or as simple as a weighting vector (for example, Adam) with the same size as ∇_{θ} resulting in a diagonal B . Considering the flow of gradients $\nabla_{\theta,t}$ over the training time as input features, the goal of preconditioning is to extract useful features that help with the updating rule. One can consider the preconditioner to be the expected value of $B(\mathbf{x}, \mathbf{y}) = [b_{ij}]^{-1}$ for

$$b_{ij} = b_{i,j}(\mathbf{x}, \mathbf{y}) = b(\nabla_{\theta_i} || \nabla_{\theta_j})^1. \quad (25)$$

In methods with a diagonal preconditioner like Adam, $B(\mathbf{x}, \mathbf{y}) = \text{diag}(\nabla_{\theta} \odot \nabla_{\theta})$, the pairwise relation between gradients is neglected. Preconditioners like Hessian inverse in Newton's method with the form of $b_{ij} = \partial \nabla_{\theta_i} / \partial \theta_j$ are based on the second derivative that encodes the cost curvature in the parameter space. In NGD and similar methods, this curvature is approximated using the second moment of gradient $b_{ij} = \nabla_{\theta_i} \nabla_{\theta_j}$, as an approximation of Hessian, in some empirical cases (see [13] for a detailed discussion).

In this section, a new preconditioning algorithm, motivated by natural gradient, is proposed for graph-based semi-supervised learning that improves the convergence of Adam and SGD with intuitive and insensitive hyper-parameters. The natural gradient is a concept from information geometry and stands for the steepest descent direction in the Riemannian manifold of probability distributions [1], where the distance in the distribution space is measured with a special *Riemannian metric*. This metric depends only on the properties of the distributions themselves and not their parameters, and in particular, it approximates the square root of the KL divergence within a small neighborhood [17]. Instead of measuring the distance between the parameters θ and θ' , the cost is measured by the KL divergence between their distributions $p(\theta)$ and $p(\theta')$. Consequently, the steepest descent direction in the statistical manifold is the negative gradient

¹Note that the adjacency matrix provides the relation between $\mathbf{x}$ samples where the preconditioning matrix includes the relation between the elements of ∇_{θ}

preconditioned with the Fisher information matrix $F(\theta)$. The validation cost on three different datasets is shown in Fig. 1 where preconditioning is applied to both Adam and SGD.

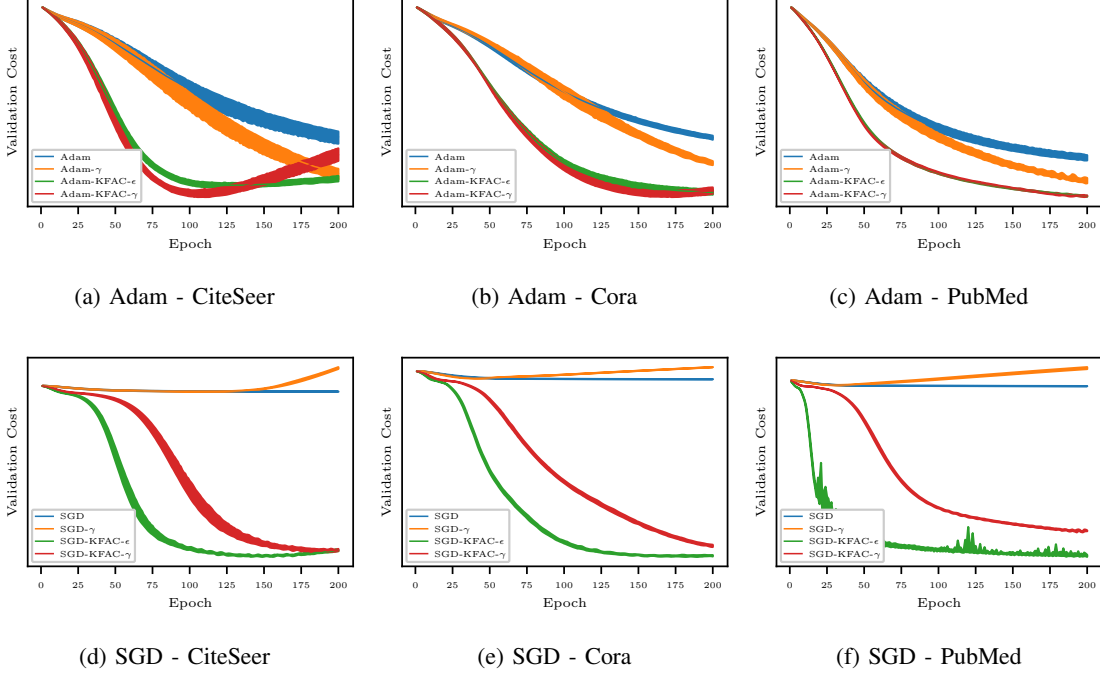


Fig. 1. The validation costs of four optimization methods on the second split of Citation datasets over 10 runs. A 2-layer GCN with a 64-dimensional hidden variable is used in all experiments. As shown in Fig. 1a, 1b, and 1c (upper row), the proposed Adam-KDAC methods (green and red curves) outperform vanilla Adam methods (blue and orange curves) on all three datasets. Also, Fig. 1d, 1e, and 1f (bottom row) reveal that the suggested SGD-KFAC methods (green and red curves) achieve a remarkably faster convergence than the vanilla SGD method (blue and orange curves) on all three datasets.

As the original NGD (Eq. 14) is defined based on a prediction function with access only to a single sample, $p(y|f(x; \theta))$, Fisher information matrix with the presence of the adjacency distribution becomes:

$$F(\theta) = E_{\mathbf{x}, \mathbf{x}', \underline{a}, y \sim p(\mathbf{x}, \mathbf{x}', \underline{a}, y; \theta)} [\nabla_{\theta} \nabla_{\theta}^T] \quad (26)$$

$$= E_{\mathbf{x}, \mathbf{x}' \sim q(\mathbf{x}), \underline{a} \sim q(\underline{a}|\mathbf{x}, \mathbf{x}'), y \sim p(y|\mathbf{x}, \mathbf{x}', \underline{a}; \theta)} [\nabla_{\theta} \nabla_{\theta}^T]. \quad (27)$$

With n samples of $q(\mathbf{x})$, i.e. X and n^2 samples of $q(\underline{a}|X)$, i.e. A , Fisher can be estimated as:

$$\hat{F}(\theta) = E_{y \sim p(y|X, A; \theta)} [\nabla_{\theta} \nabla_{\theta}^T], \quad (28)$$

where

$$\nabla_{\theta} = -\nabla_{\theta} \log p(X, A, y; \theta). \quad (29)$$

In fact, to evaluate the expectation in Eq. 26, $q(X)$ and $q(A|X)$ are approximated with $\hat{q}(X)$ and $\hat{q}(A|X)$ using $\{\mathbf{x}_i\}_{i=1}^n$ and A , respectively. However, there are only $\bar{n}$ samples from $\hat{q}(\mathbf{y}|\mathbf{x}_j)$ as an approximation of $q(\mathbf{y}|\mathbf{x}_j)$ for the following replacement:

$$p(\mathbf{y}|X, A; \boldsymbol{\theta}) \approx \hat{q}(\mathbf{y}|\mathbf{x}_i). \quad (30)$$

Therefore, an empirical Fisher can be obtained by

$$\hat{F}(\boldsymbol{\theta}) = \frac{1}{\bar{n}} \sum_{i=1}^{\bar{n}} \nabla_{\boldsymbol{\theta},i} \nabla_{\boldsymbol{\theta},i}^T = \sum_{i=1}^{\bar{n}} B_i(\boldsymbol{\theta}) \quad (31)$$

for

$$\nabla_{\boldsymbol{\theta},i} = -\nabla_{\boldsymbol{\theta}} \log p(\mathbf{y}_i|X, A; \boldsymbol{\theta}) \quad (32)$$

$$B_i(\boldsymbol{\theta}) = B(X, A, \mathbf{y}_i; \boldsymbol{\theta}). \quad (33)$$

From the computation perspective, the matrix $B_i(\boldsymbol{\theta})$ can be very large, for example, in neural networks with multiple layers, the parameters could be huge, so it needs to be approximated too. In networks characterized with Eqs. 17 or 20, a simple solution would be ignoring the cross-layer terms so that $B_i(\boldsymbol{\theta})^{-1}$ and consequently $B_i(\boldsymbol{\theta})$ turns into a block-diagonal matrix:

$$B_i(\boldsymbol{\theta}) = \text{diag}(B_{1,i}, \dots, B_{m,i}) \quad (34)$$

In KFAC, the diagonal block $B_{k,i}$, corresponded to k 'th layer with the dimension $d_k d_{k-1} \times d_k d_{k-1}$, is approximated with the Kronecker product of the inverse of two smaller matrices $U_{k,i}$ and $V_{k,i}$ as:

$$B_{k,i} = (U_{k,i} \otimes V_{k,i})^{-1} = U_{k,i}^{-1} \otimes V_{k,i}^{-1}. \quad (35)$$

For $\nabla_{\boldsymbol{\theta},i} = [\nabla_{\boldsymbol{\theta}_1,i}^T, \dots, \nabla_{\boldsymbol{\theta}_m,i}^T]^T$, the preconditioned gradient $B_{k,i} \nabla_{\boldsymbol{\theta}_k,i}$ can be computed using the identity

$$B_{k,i} \nabla_{\boldsymbol{\theta}_k,i} = U_{k,i}^{-1} \otimes V_{k,i}^{-1} \text{vec}\left(\frac{\partial l}{\partial W_k}\right) \quad (36)$$

$$= \text{vec}(U_{k,i}^{-1} \frac{\partial l}{\partial W_k} V_{k,i}^{-1}). \quad (37)$$

Noting that:

$$\frac{\partial l}{\partial W_k} = \left(\frac{\partial l}{\partial \mathbf{x}_k} \odot \phi'_k(W_k \tilde{\mathbf{x}}_{k-1}) \right) \tilde{\mathbf{x}}_{k-1}^T \quad (38)$$

$$= \mathbf{u}_{k,i} \mathbf{v}_{k,i}^T, \quad (39)$$

U_k and V_k blocks are approximated with the expected values of $\mathbf{u}_{k,i}\mathbf{u}_{k,i}^\top$ and $\mathbf{v}_{k,i}\mathbf{v}_{k,i}^\top$ respectively where $\dim(\mathbf{u}_k) = d_k$, $\dim(\mathbf{v}_k) = d_{k-1}$. Finally, U_k^{-1} and V_k^{-1} are evaluated by taking inverses of $U_k + \epsilon^{-1/2}$ and $V_k + \epsilon^{-1/2}$ for ϵ being the regularization hyper-parameter.

For a graph with n nodes, adjacency matrix A , and the training set $\{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^{\bar{n}} + \{\mathbf{x}_i\}_{i=\bar{n}+1}^n$, U_k and V_k are estimated in two ways: (1) using only $\bar{n}$ labeled samples, and (2) including $n - \bar{n}$ unlabeled samples. In the first method, U_k and V_k are estimated by:

$$U_k = \frac{1}{\bar{n}} \left(\frac{\partial l}{\partial X_k} \odot \phi'_k(W_k \tilde{X}_{k-1}) \right) \left(\frac{\partial l}{\partial X_k} \odot \phi'_k(W_k \tilde{X}_{k-1}) \right)^\top \quad (40)$$

$$V_k = \frac{1}{\bar{n}} \tilde{X}_{k-1} \tilde{X}_{k-1}^\top. \quad (41)$$

Note that both $\partial l / \partial X_k$ and $\phi'_k(W_k \tilde{X}_{k-1})$ are $d_k \times n$ matrices and the last $n - \bar{n}$ columns of $\partial l / \partial X_k$ are zero. However, as unlabeled samples are not used in the first method, one needs to evaluate loss function for $i = \bar{n} + 1, \dots, n$, which can be done by sampling $\hat{\mathbf{y}}_i$ from $p(\mathbf{y}|\mathbf{x}; \boldsymbol{\theta})$. In the second method, these new samples are added to the empirical cost as

$$\begin{aligned} \hat{r}(\boldsymbol{\theta}) &= \frac{1}{\bar{n}} \sum_{i=1}^{\bar{n}} l(\mathbf{y}_i, f(X, A; \boldsymbol{\theta})) \\ &\quad + \frac{\lambda}{n - \bar{n}} \sum_{i=\bar{n}+1}^n l(\mathbf{y}_i, f(X, A; \boldsymbol{\theta})), \end{aligned} \quad (42)$$

where $0 \leq \lambda \leq 1$ denotes the regularization hyper-parameter for controlling the cost of predicted labels and $\lambda = 0$ results the first method. As the prediction improves over the course of training, λ can be a function of iteration t , for example here, it is defined to be:

$$\lambda(t) := \left(\frac{t}{\max(t)} \right)^\gamma, \quad (43)$$

where $\max(t)$ shows the maximum number of iterations and γ is the replaced regularization hyper-parameter. Algorithm 1 shows the preconditioning step for modifying gradients of each layer at any iteration such that gradients are first, transformed using two matrices of V_k^{-1} and U_k^{-1} , then sent to the optimization algorithm for updating parameters.

A. Relation between Fisher and Hessian

The Hessian of the cost function:

$$H_{\boldsymbol{\theta}} r(\boldsymbol{\theta}) = E_{X, A, \mathbf{y} \sim p(X, A, \mathbf{y}; \boldsymbol{\theta})} [H_{\boldsymbol{\theta}} l(\mathbf{y}, f(X, A; \boldsymbol{\theta}))] \quad (44)$$

Algorithm 1 Semi-Supervised Preconditioning

Require: ∇W_k ▷ Gradient of parameters for $k = 1, \dots, m$
Require: A ▷ Adjacency matrix
Require: D ▷ Degree matrix
Require: $\mathbf{z}$ ▷ Training mask vector
Require: ϵ, λ ▷ Regularization hyper-parameters

$$n = \dim(\mathbf{z})$$

$$\bar{n} = \sum(\mathbf{z})$$

$$\tilde{A} = (D + I)^{-1/2}(A + I)(D + I)^{-1/2} = [\tilde{a}_{ij}]$$

for $k = 1, \dots, m$ **do**

$$\tilde{\mathbf{x}}_{k-1,i} = \sum_{j=1}^n \tilde{a}_{i,j} \mathbf{x}_{k-1,j}$$

$$\mathbf{u}_{k-1,i} = \partial l / \partial \mathbf{x}_k \odot \phi'_k(W_k \tilde{\mathbf{x}}_{k-1,i})$$

$$\mathbf{v}_{k-1,i} = \tilde{\mathbf{x}}_{k-1,i}$$

$$U_k = \sum_{i=1}^n (z_i + (1 - z_i)\lambda) \mathbf{u}_{k-1,i} \mathbf{u}_{k-1,i}^\top / (n + \lambda \bar{n})$$

$$V_k = \sum_{i=1}^n (z_i + (1 - z_i)\lambda) \mathbf{v}_{k-1,i} \mathbf{v}_{k-1,i}^\top / (n + \lambda \bar{n})$$

$$U_k^{-1} = \text{INVERSE}(U_k)$$

$$V_k^{-1} = \text{INVERSE}(V_k)$$

output $V_k^{-1} \nabla W_k U_k^{-1}$

function INVERSE(X)

output $(X + \epsilon^{-1/2} I)^{-1}$

can also be approximated using $\hat{q}(X)$, $\hat{q}(A|X)$, and $\hat{q}(\mathbf{y}|\mathbf{x}_i)$ resulting the empirical Hessian to be

$$\hat{H}_{\theta r}(\boldsymbol{\theta}) := \frac{1}{\bar{n}} \sum_{i=1}^{\bar{n}} H_{\theta} l(\mathbf{y}_i, f(X, A; \boldsymbol{\theta})), \quad (45)$$

which is equivalent to the empirical Fisher Eq. 31 when $p(X, A, \mathbf{y}; \boldsymbol{\theta})$ is estimated with $\hat{q}(X)\hat{q}(A|X)\hat{q}(\mathbf{y}|\mathbf{x}_i)$ for $i = 1, \dots, \bar{n}$ (see Lemma 1 in the appendix).

IV. EXPERIMENTS

In this section, the performance of the proposed algorithm is evaluated compared to Adam and SGD on several datasets for the task of node classification in single graphs. The task is assumed to be transductive when all the features are available for training but only a portion of labels are used in the training. First, a detailed description of datasets and the model architecture are provided. Then, the general optimization setup, commonly used for the node classification, is specified. The last part includes the sensitivity to hyper-parameter and training time analysis in addition to validation cost convergence and the test accuracy. All the experiments are conducted mainly using Pytorch [19] and Pytorch Geometric [5], two open-source Python libraries for automating differentiation and working with graph datasets.

A. Datasets

Three citation datasets with the statistics shown in Table II are used in the experiments [21]. Cora, CiteSeer, and PubMed are single graphs in which nodes and edges correspond to documents and citation links, respectively. A sparse feature vector (document keywords) and a class label are associated with each node. Several splits of these datasets are used in the node classification task. The first split, 20 instances are randomly selected for training, 500 for validation, and 1000 for the test; the rest of the labels are not used [23]. In the second split, all nodes except 500+1000 validation and test nodes are used for the training [3]. To evaluate the overfitting behavior, the third split exploits all labels for training excluding 500 + 500 nodes for the validation and test [15].

TABLE II
CITATION NETWORK DATASETS STATISTICS

Dataset	Nodes	Edges	Classes	Features
Citeseer	3,327	4732	6	3,703
Cora	2,708	5,429	7	1,433
Pubmed	19,717	44,338	3	500

B. Architectures

In the node classification using a NN followed by Softmax function (Eq. 5), the class with maximum probability is chosen to be the predicted node label. A 2-layer GCN with a 64-dimensional hidden variable is used for comparing different optimization methods. In the first

layer, the activation function ReLU is followed by a dropout function with a rate of 0.5. The loss function is evaluated as the negative log-likelihood of Softmax (Eq. 5) of the last layer.

C. Optimization

The weights of parameters are initialized like the original GCN [10] and input vectors are row-normalized accordingly [7]. The model is trained for 200 epochs without any early stopping and the learning rate of 0.01. The Adam and SGD are used with the weight decay of 5×10^{-4} and the momentum of 0.9, respectively.

D. Results

The optimization performance is measured by both the minimum validation cost and the test accuracy for the best validation cost. The validation cost of training a 2-layer GCN with a 64-dimensional hidden variable is used for comparing optimization methods (Adam and SGD) with their preconditioned version (Adam-KFAC and SGD-KFAC). For each method, unlabeled samples are used in the training process with a ratio controlled by γ . Fig. 1 shows the validation cost of four methods based on Adam (upper row) and SGD (bottom row) for all three Citation datasets. The test accuracy of a 2-layer GCN trained using four different methods on three split are shown in Tab. III, IV, and V. Reported values of test accuracy in tables are averages and 95% confidence intervals over 10 runs for the best hyper-parameters tuned on the second split of the CiteSeer dataset. Note that the test accuracy may not always reflect the performance of the optimization method as the objective function (cross-entropy) is not the same as the prediction function (argmax). However, in most cases, the proposed method achieves better accuracy compared to Adam (the first row in all tables). As a fixed learning rate 0.01 is used in all methods, SGD has a very slow convergence and does not provide competitive results.

The importance of hyper-parameters ϵ , γ are shown in Fig. 2. Figures 2a and 2d depict the sensitivity of Adam and SGD to the ϵ parameter, respectively. As the inverse of ϵ directly affects the same factor as the learning rate η , the smaller the ϵ , the faster the convergence. However, choosing very small ϵ results in larger confidence intervals which are not desirable. The effect of γ on Adam and SGD are depicted in figures 2b and 2e, respectively. When using Adam, due to its faster convergence compared to SGD, smaller γ , i.e. using more predictions leads to much wider confidence intervals. In other words, the training process dominated by more labels results in a more stable convergence with a smaller variance. Thus, for a stable estimation, λ or γ must be tuned with respect to the optimization algorithm because of their sensitivity to

TABLE III

THE TEST ACCURACY OF FOUR OPTIMIZATION METHODS ON THE FIRST SPLIT OF CITATION DATASETS OVER 10 RUNS. A 2-LAYER GCN WITH A 64-DIMENSIONAL HIDDEN VARIABLE IS USED IN ALL EXPERIMENTS.

	CiteSeer	Cora	Pubmed
Adam	71.66 ± 0.61	81.20 ± 0.25	79.72 ± 0.30
Adam _{γ}	74.28 ± 0.67	82.42 ± 0.33	80.06 ± 0.34
Adam-KFAC _{ϵ}	71.94 ± 0.53	81.68 ± 0.25	79.48 ± 0.28
Adam-KFAC _{γ}	70.24 ± 0.66	82.84 ± 0.87	76.94 ± 0.59
SGD	20.38 ± 8.92	23.14 ± 5.17	45.76 ± 3.04
SGD _{γ}	17.64 ± 6.18	17.26 ± 8.41	46.20 ± 4.35
SGD-KFAC _{ϵ}	71.82 ± 0.48	82.06 ± 0.34	77.20 ± 0.63
SGD-KFAC _{γ}	73.52 ± 0.22	81.70 ± 0.79	79.20 ± 0.29

TABLE IV

THE TEST ACCURACY OF FOUR OPTIMIZATION METHODS ON THE SECOND SPLIT OF CITATION DATASETS OVER 10 RUNS. A 2-LAYER GCN WITH A 64-DIMENSIONAL HIDDEN VARIABLE IS USED IN ALL EXPERIMENTS.

	CiteSeer	Cora	Pubmed
Adam	78.68 ± 0.83	87.36 ± 0.47	87.78 ± 0.14
Adam _{γ}	77.98 ± 0.39	87.28 ± 0.34	87.52 ± 0.30
Adam-KFAC _{ϵ}	79.50 ± 0.15	87.60 ± 0.20	88.46 ± 0.24
Adam-KFAC _{γ}	79.42 ± 0.32	86.60 ± 0.30	87.88 ± 0.16
SGD	20.80 ± 2.12	31.90 ± 0.00	43.22 ± 1.42
SGD _{γ}	20.96 ± 5.22	31.90 ± 0.00	40.82 ± 0.33
SGD-KFAC _{ϵ}	79.48 ± 0.40	87.54 ± 0.43	89.08 ± 0.18
SGD-KFAC _{γ}	77.32 ± 0.27	87.42 ± 0.24	88.18 ± 0.30

TABLE V

THE TEST ACCURACY OF FOUR OPTIMIZATION METHODS ON THE THIRD SPLIT OF CITATION DATASETS OVER 10 RUNS. A 2-LAYER GCN WITH A 64-DIMENSIONAL HIDDEN VARIABLE IS USED IN ALL EXPERIMENTS.

	CiteSeer	Cora	Pubmed
Adam	79.80 ± 0.66	89.44 ± 0.41	87.16 ± 0.71
Adam _{γ}	79.64 ± 0.32	89.60 ± 0.91	87.44 ± 0.27
Adam-KFAC _{ϵ}	80.52 ± 0.14	90.16 ± 0.59	87.84 ± 0.21
Adam-KFAC _{γ}	80.52 ± 0.22	89.24 ± 0.64	87.36 ± 0.37
SGD	15.04 ± 1.70	32.80 ± 0.00	41.96 ± 0.44
SGD _{γ}	16.12 ± 5.30	32.80 ± 0.00	41.20 ± 0.00
SGD-KFAC _{ϵ}	79.76 ± 0.75	89.88 ± 0.14	89.36 ± 0.57
SGD-KFAC _{γ}	78.52 ± 0.28	88.72 ± 0.38	87.88 ± 0.80

the convergence rate. Since the Fisher matrix does not change considerably at each iteration, an experiment is performed to explore the sensitivity of validation loss to the frequency of updating Fisher. In Figures 2c and 2f, the validation cost over time is evaluated for updating Fisher every 4, 8, $\dots$, 128 iterations. When Fisher is updated more frequently, its computation takes more time hence the training process lasts longer (having other hyper-parameters fixed). Increasing the update frequency does not affect the performance to some extent, however, it largely reduces the training time. As updating Fisher every 50 or 100 iterations, does not affect the final validation cost to a great extent, to speed up the training process, Fisher is updated every 50 epochs in all of the experiments.

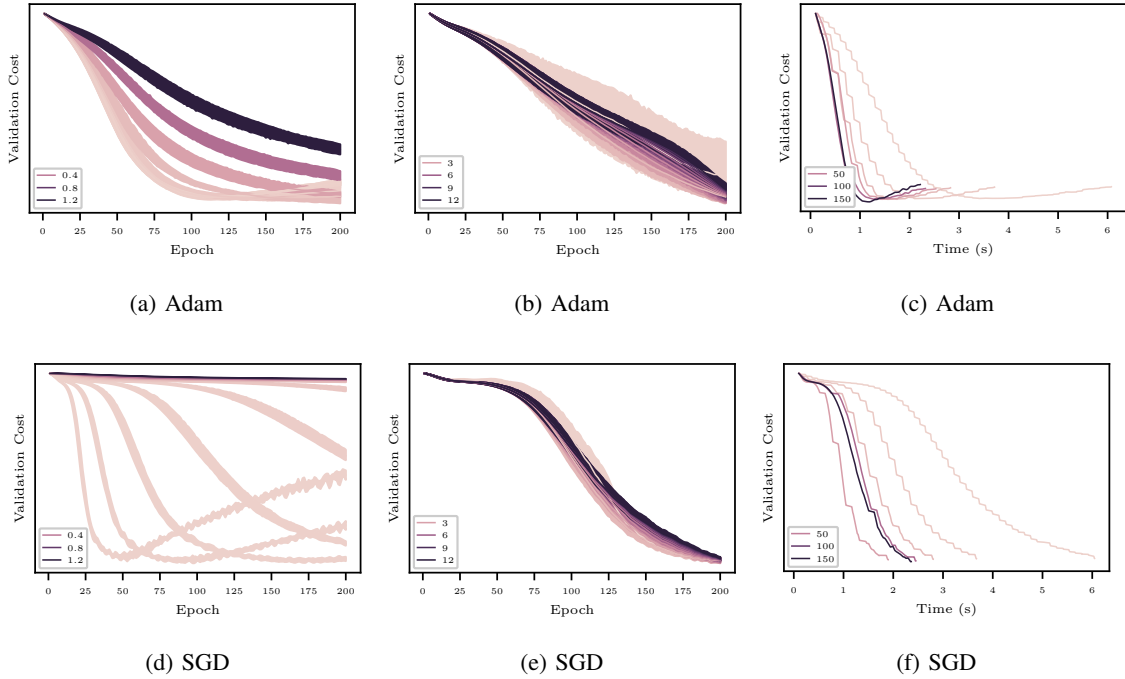


Fig. 2. The sensitivity of ϵ , γ , and updating frequency on validation costs of Adam-KFAC (upper) and SGD-KFAC (below) when training on the second split of CiteSeer dataset over 10 runs. A 2-layer GCN with a 64-dimensional hidden variable is used in all experiments. Fig. 2a and 2d show that smaller ϵ results in a faster convergence with a probable cost of larger variance as it inversely scales the same factor as the learning rate. As depicted in Fig. 2b and 2e, the larger the γ , the more stable the convergence (the more confined confidence intervals). Finally, it can be seen in Fig. 2c and 2f that since performances are similar under different updating frequencies, selecting a relatively large frequency (50) can reduce the training time substantially.

To examine the time complexity of the proposed method, the validation costs of Adam-KFAC and SGD-KFAC are compared with Adam and SGD when training on the second split of Citation datasets with respect to the training time for 200 epochs (Fig. 3). The training on

Cora and PubMed (Fig. 3b and 3c) takes a shorter time compared to the training on CitSeer (Fig. 3a) mainly because of the dimension of input features as it directly enlarges the size of the Fisher matrix. As shown in Fig. 3, the proposed SGD-KFAC method (red curve) converges much faster than the vanilla SGD as expected. Surprisingly, SGD-KFAC outperforms Adam and even Adam-KFAC methods in all datasets implying that the naive SGD with a natural gradient preconditioner can lead to a faster convergence than Adam-based methods. Another interesting observation is that Adam-based methods demonstrate similar performances in all experiments making them independent of the dataset while SGD-based methods show different overfitting behavior.

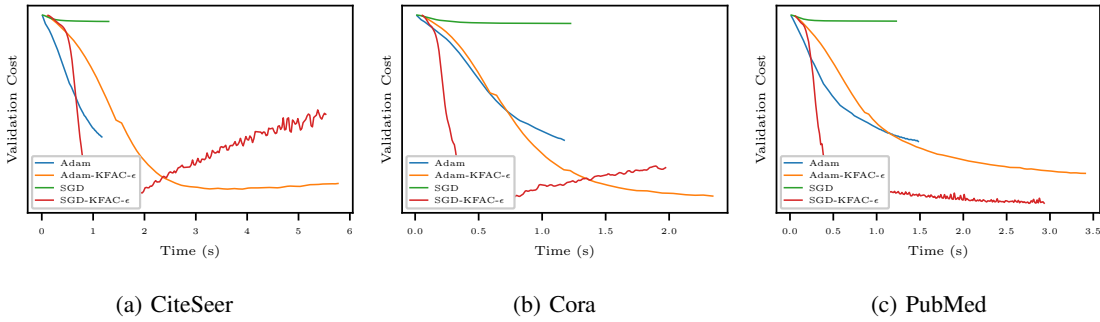


Fig. 3. The validation costs of four optimization methods with respect to the training time on the second split of Citation datasets over 10 runs. A 2-layer GCN with a 64-dimensional hidden variable is used in all experiments. The proposed SGD-KFAC method shows the highest convergence rate among all other methods and it is slightly faster than Adam-KFAC.

V. CONCLUSION

In this work, we introduced a novel optimization framework for graph-based semi-supervised learning. After the distinct definition of semi-supervised problems with the adjacency distribution, we provided a comprehensive review of topics like semi-supervised learning, graph neural network, and preconditioning optimization (and NGD as its especial case). We adopted a commonly used probabilistic framework covering least-squared regression and cross-entropy classification. In the node classification task, our proposed method showed to improve Adam and SGD not only in the validation cost but also in the test accuracy of GCN on three splits of Citation datasets. Extensive experiments were provided on the sensitivity to hyper-parameters and the time complexity. As the first work, to the best of our knowledge, on the preconditioned optimization of graph neural networks, we not only achieved the best test accuracy but also empirically showed that it can be used with both Adam and SGD.

As the preconditioner may significantly affect Adam, illustrating the relation between NGD and Adam and effectively combining them can be a promising direction for future work. We also aim to deploy faster approximation methods than KFAC like [6] and better sampling methods for exploiting unlabeled samples. Finally, since this work is mainly focused on single parameter layers, another possible research path would be adjusting KFAC to, for example, residual layers [8].

ACKNOWLEDGMENT

YF and LL were partially supported by NSF grants DMS Career 1654579 and DMS 1854779.

REFERENCES

- [1] Shun-Ichi Amari. “Natural gradient works efficiently in learning”. In: *Neural computation* 10.2 (1998), pp. 251–276.
- [2] Michael M Bronstein et al. “Geometric deep learning: going beyond euclidean data”. In: *IEEE Signal Processing Magazine* 34.4 (2017), pp. 18–42.
- [3] Jie Chen, Tengfei Ma, and Cao Xiao. “Fastgcn: fast learning with graph convolutional networks via importance sampling”. In: *arXiv preprint arXiv:1801.10247* (2018).
- [4] John Duchi, Elad Hazan, and Yoram Singer. “Adaptive subgradient methods for online learning and stochastic optimization.” In: *Journal of machine learning research* 12.7 (2011).
- [5] Matthias Fey and Jan Eric Lenssen. “Fast graph representation learning with PyTorch Geometric”. In: *arXiv preprint arXiv:1903.02428* (2019).
- [6] Thomas George et al. “Fast approximate natural gradient descent in a kronecker factored eigenbasis”. In: *Advances in Neural Information Processing Systems*. 2018, pp. 9550–9560.
- [7] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feed-forward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256.
- [8] Kaiming He et al. “Deep residual learning for image recognition”. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.
- [9] Diederik P Kingma and Jimmy Ba. “Adam: A method for stochastic optimization”. In: *arXiv preprint arXiv:1412.6980* (2014).

- [10] Thomas N Kipf and Max Welling. “Semi-supervised classification with graph convolutional networks”. In: *arXiv preprint arXiv:1609.02907* (2016).
- [11] Alexander Kolesnikov, Xiaohua Zhai, and Lucas Beyer. “Revisiting self-supervised visual representation learning”. In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2019, pp. 1920–1929.
- [12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “Imagenet classification with deep convolutional neural networks”. In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.
- [13] Frederik Kunstner, Philipp Hennig, and Lukas Balles. “Limitations of the empirical Fisher approximation for natural gradient descent”. In: *Advances in Neural Information Processing Systems*. 2019, pp. 4156–4167.
- [14] Yann LeCun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.
- [15] Ron Levie et al. “Cayleynets: Graph convolutional neural networks with complex rational spectral filters”. In: *IEEE Transactions on Signal Processing* 67.1 (2018), pp. 97–109.
- [16] Alexander Ly et al. “A tutorial on Fisher information”. In: *Journal of Mathematical Psychology* 80 (2017), pp. 40–55.
- [17] James Martens. “New insights and perspectives on the natural gradient method”. In: *arXiv preprint arXiv:1412.1193* (2014).
- [18] James Martens and Roger Grosse. “Optimizing neural networks with kronecker-factored approximate curvature”. In: *International conference on machine learning*. 2015, pp. 2408–2417.
- [19] Adam Paszke et al. “Pytorch: An imperative style, high-performance deep learning library”. In: *Advances in neural information processing systems*. 2019, pp. 8026–8037.
- [20] Franco Scarselli et al. “The graph neural network model”. In: *IEEE Transactions on Neural Networks* 20.1 (2008), pp. 61–80.
- [21] Prithviraj Sen et al. “Collective classification in network data”. In: *AI magazine* 29.3 (2008), pp. 93–93.
- [22] Zonghan Wu et al. “A comprehensive survey on graph neural networks”. In: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

- [23] Zhilin Yang, William Cohen, and Ruslan Salakhudinov. “Revisiting semi-supervised learning with graph embeddings”. In: *International conference on machine learning*. 2016, pp. 40–48.
- [24] Jie Zhou et al. “Graph neural networks: A review of methods and applications”. In: *arXiv preprint arXiv:1812.08434* (2018).

APPENDIX

Lemma 1. *The expected value of the Hessian of $-\log p(X, A, \mathbf{y}; \boldsymbol{\theta})$ is equal to Fisher information matrix, i.e.*

$$-E_{X,A,\mathbf{y} \sim p(X,A,\mathbf{y};\boldsymbol{\theta})}[H_{\boldsymbol{\theta}} \log p(X, A, \mathbf{y}; \boldsymbol{\theta})] = F \quad (46)$$

Proof. The Hessian of $f(\boldsymbol{\theta})$ can be written as the Jacobian of $\nabla_{\boldsymbol{\theta}} f$:

$$H_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = J_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}). \quad (47)$$

So for the Hessian of the negative log-likelihood becomes:

$$-H_{\boldsymbol{\theta}} \log p(X, A, \mathbf{y}; \boldsymbol{\theta}) \quad (48)$$

$$= -J_{\boldsymbol{\theta}} \frac{\nabla_{\boldsymbol{\theta}} p(X, A, \mathbf{y}; \boldsymbol{\theta})}{p(X, A, \mathbf{y}; \boldsymbol{\theta})} \quad (49)$$

$$= -\frac{H_{\boldsymbol{\theta}} p(X, A, \mathbf{y}; \boldsymbol{\theta}) \cdot p(X, A, \mathbf{y}; \boldsymbol{\theta})}{p(X, A, \mathbf{y}; \boldsymbol{\theta}) \cdot p(X, A, \mathbf{y}; \boldsymbol{\theta})} \quad (50)$$

$$- \frac{\nabla_{\boldsymbol{\theta}} p(X, A, \mathbf{y}; \boldsymbol{\theta}) \nabla_{\boldsymbol{\theta}} p(X, A, \mathbf{y}; \boldsymbol{\theta})^{\top}}{p(X, A, \mathbf{y}; \boldsymbol{\theta}) \cdot p(X, A, \mathbf{y}; \boldsymbol{\theta})} \quad (51)$$

$$= -\frac{H_{\boldsymbol{\theta}} p(X, A, \mathbf{y}; \boldsymbol{\theta})}{p(X, A, \mathbf{y}; \boldsymbol{\theta})} + \nabla_{\boldsymbol{\theta}} \nabla_{\boldsymbol{\theta}}^{\top} \quad (52)$$

Taking the expectation over $p(X, A, \mathbf{y}; \boldsymbol{\theta})$, the first term turns into zero:

$$E_{X,A,\mathbf{y} \sim p(X,A,\mathbf{y};\boldsymbol{\theta})} \left[\frac{H_{\boldsymbol{\theta}} p(X, A, \mathbf{y}; \boldsymbol{\theta})}{p(X, A, \mathbf{y}; \boldsymbol{\theta})} \right] \quad (53)$$

$$= H_{\boldsymbol{\theta}} E_{X,A,\mathbf{y} \sim p(X,A,\mathbf{y};\boldsymbol{\theta})} [1] \quad (54)$$

$$= 0 \quad (55)$$

and Fisher is defined as the expected value of the second term. $\square$