GRAPH OPTIMIZATION IN NEURAL MODELS

A Project

Report

Submitted in the partial fulfillment of the requirements for

the award of the degree of

Bachelor of Technology

in

Department of Computer Science Engineering

by

K. Rushmitha - 180030031

V. Mounica - 180030075

T. V.Lokesh Sai Kumar Reddy -180030318

under the supervision of

Dr P.Vithya Ganasan

Department of Computer Science Engineering

K L E F, Green Fields,

Vaddeswaram- 522502, Guntur(Dist), Andhra Pradesh, India.

November, 2021

DECLARATION

The Project Report entitled “Graph Optimization in Neural Models” is a record of bonafide work of K. Rushmitha – 180030031 and V. Mounica- 180030075 and T.V. Lokesh Sai Kumar Reddy- 180030318, submitted in partial fulfillment for the award of B.Tech in Computer Science Engineering to the K L University. The results embodied in this report have not been copied from any other departments/University/Institute..

K. Rushmitha- 180030031

V. Mounica - 180030075

T. V.LokeshSai Kumar Reddy - 180030318

CERTIFICATE

This is to certify that the Project Report entitled "Graph Optimization in Neural Models" is being submitted by K. Rushmitha – 180030031 and V. Mounica- 180030075 and T.V. Lokesh Sai Kumar Reddy- 180030318 submitted in partial fulfillment for the award of B.Tech in Computer Science Engineering to the K L University is a record of bonafide work carried out under our guidance and supervision.The results embodied in this report have not been copied from any other departments/ University/Institute.

Signature of the Supervisor

Dr.P.Vithya Ganasan

Signature of the HOD

Signature of the External Examiner

ACKNOWLEDGEMENTS

We are very thankful to our project guide Dr.P. Vithya Ganasan for her continuous support in completing the project work. Without her encouragement and help the project could not have been completed.

We take immense pleasure in expressing our gratitude to our head of department Mr.Hari KiranVege, Head of the Department (CSE) for his priceless words of encouragement without which this would have been a herculean task.

We express our heartful gratitude to our president, Chancellor, Vice Chancellor, and our beloved principal Dr. K. SubbaRao for providing infrastructure facilities in pursuing this project.

Finally, we thank all Teaching and Non-Teaching staff of our department and especially my classmates and my friends for their support in the completion of our project work.

K. Rushmitha -180030031

V. Mounica - 180030075

T. V.Lokesh Sai Kumar Reddy - 180030318

ABSTRACT

Machine learning in today's era is influencing various different features on various different products. So far this paradigm has been restricted to the cloud and with increasing capability on mobile devices things are moving towards on device ML so optimization became the need of hour for onDevice Execution. With the growth of different model frameworks like TensorFlow.Caffeetc , the expansion and application of models has increased.

It also leads to cross device and cross framework usage of same model. Onnx helps in achieving the same. However, optimizations so far has been device specific which can be generalized to Onnx framework. The existing optimizations device specific can be analyzed and optimized so it can be applied to different model types and CPU/GPU etc.It is also required to see the impact on Accuracy due to the generalized optimization done or evaluation on different types of optimization techniques comparison to choose the best one.

CONTENTS

CHAPTER 1:

INTRODUCTION

1.1) Graph and Optimization

An usual graph is described as collection of vertices(points) and edges(lines) between the edges joining the vertices. Graph consists of variety kinds to be mentioned like directed and undirected graphs or weighted graphs or simple graphs. We could have more information attached to our vertices or our edges.

Optimization is particular approach to solving problems. In optimization we solve problems by specifying them in terms of what are the valid solutions and what are the costs of solutions so we need a cost function that takes us from the space of valid solutions to a cost for each solution and then our goal is just to find the solution that has minimum cost amongst the valid solutions.

1.2) Graph → Optimization

Graph is, at its core, a study of relationships. Given a collection of nodes and connections that can abstract anything from city plans to computer data, graph theory is a valuable tool for quantifying and simplifying the numerous moving features of dynamic systems. Many arrangement, networking, optimization, matching, and operational problems can be solved by studying graphs using a framework. Graphs can be used to model a wide range of linkages and processes in physical, biological, sociological, and information systems, and they have many applications.

Now that we have now persuaded you that graph is worth learning about, let's look at our example of route planning when collecting products in our factory.

Challenge :

The task at hand is to identify the quickest route that passes all of the pickup places while also adhering to the limits on where it is possible to drive, given a "selection list" as input. Crossing between factory corridors is only permitted at certain "key events," according to the assumptions and limits. In addition, the travel direction must adhere to each corridor's permissible driving direction.

Solution :

In the graph, all pickup points in the factory form a "node," with the edges representing permitted lanes/corridors and distances between the nodes. To more officially introduce the topic, let's start with a basic example.

The graph below depicts two corridors, each with five storage points. Every block is represented as a node in the graph, with an address ranging from 1 to 10. The arrows indicate the allowed driving direction, whereas the double arrows indicate that you can drive in either direction.

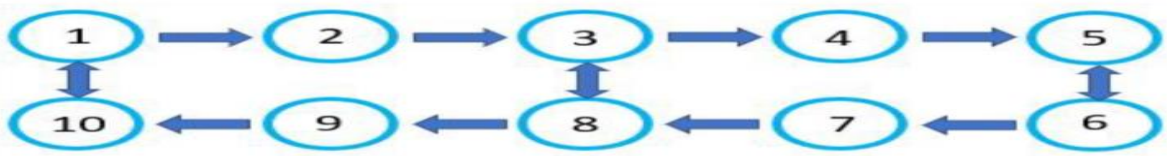


Fig 1.1 : Basic Graph Representation

Because we can represent the permitted driving routes as a graph, we can use graph theory mathematical techniques to find the best "mobility route" between the nodes.

An adjacency matrix can be used to quantitatively characterise the sample graph above. As a result, the adjacency matrix on the right in the diagram below represents our factory graph, indicating all legal driving paths between the various nodes.

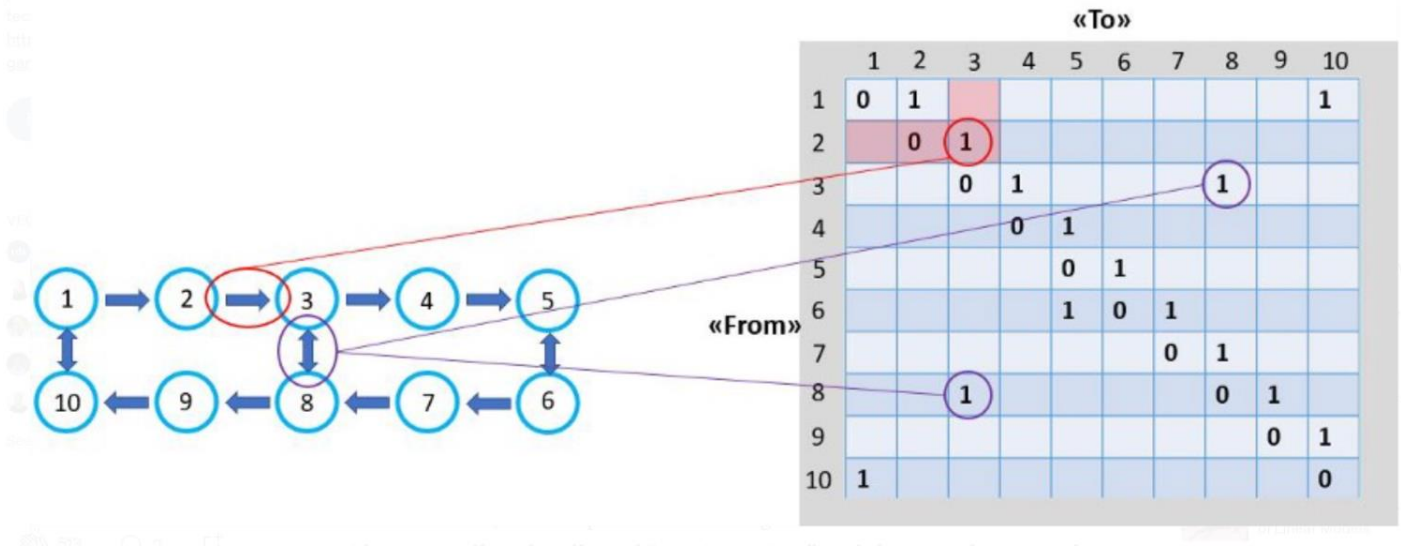


Fig 1.2 : Optimized graph

Example : We are permitted to move between nodes 2 and 3, but not through nodes 3 and 2. The "1" in the adjacency matrix on the right indicates this.

1.3) Frameworks

Machine learning is a difficult field to master. Machine learning frameworks, on the other hand, have made data collection much easier, training models, serving predictions, and improving future outcomes far less

intimidating and complex than it used to be, deploying machine learning models is far less frightening and difficult than it used to be.

i) Tensorflow

TensorFlow is an open source library for large-scale machine learning and numerical computing. TensorFlow combines a variety of machine learning and deep learning models and algorithms into a single metaphor that makes them practical. It makes use of Python to create a user-friendly front-end API for developing applications with the framework, which is then executed in high-performance C++.

Dataflow graphs—structures that explain how data passes through a graph, or a sequence of processing nodes—can be created with TensorFlow. Each node in the graph symbolises a mathematical process, and each node-to-node connection or edge is a tensor, or multidimensional data array. TensorFlow applications can be launched on almost any device, including a local PC, a cloud cluster, iOS and Android smartphones, CPUs, and GPUs. If you use Google's own cloud, you can accelerate TensorFlow by running it on Google's proprietary TensorFlow Processing Unit (TPU) silicon.

TensorFlow's models, on the other hand, may be used to solve issues on virtually any device.

ii)Caffe

Caffe is a deep learning framework that emphasizes flexibility, performance, and modularity. It's being worked on by Berkeley AI Research (BAIR) and community contributors. During his PhD at UC Berkeley, YangqingJia developed the project.

Caffe models are machine learning engines that work from start to finish. The net is a collection of layers connected by a computing graph - specifically, a directed acyclic graph (DAG). Caffe handles all of the accounting for any DAG of layers, ensuring that the forward and backward passes are accurate.

iii)ONNX

ONNX is a machine learning framework that acts as a translator between other machine learning frameworks. So, imagine you're working with TensorFlow and want to move on to TensorRT, or you're working with PyTorch and want to move on to TFLite or another machine learning framework. ONNX is a fantastic tool for converting your model as you move through these many machine learning frameworks.

So ONNX has put in a lot of effort to basically integrate all types of different neural network functions and functionalities in these machine learning models, so I can support this cross functionality and have a common framework to convert into.ONNX is supported by a variety of platforms (snapdragon, neural processing engine) and can be used to evaluate neural networks on mobile devices.It can be used to perform inference between two Android apps without requiring the use of cloud services.

ONNX Runtime has a number of graph improvements to help you increase your model's speed. Graph optimizations are transformations at the graph level, ranging from simple graph simplifications and node eliminations to more complicated node fusions and layout improvements.

The Task Graph The process of transferring a task graph to a destination platform is known as scheduling. The task graph represents the application: Edges reflect task precedence limitations, while nodes represent computational jobs. Each task is given an assignment (the processor that will perform the work) and a schedule (the time when the execution will begin). The goal is to achieve an efficient execution of the application by optimising some objective function, most often the overall execution time.

1.4) Optimization techniques

Optimization strategies are a set of sophisticated instruments for effectively managing an enterprise's resources and, as a result, maximising shareholder profit.

1) Graph Rewriting

Graph rewriting is the computational process of producing a new graph from an existing one. It has a wide range of applications, from software engineering to layout algorithms to image production. It may also be used as a computation abstraction.The basic idea is that the state of a computation can be represented as a graph, and subsequent steps in that computation can be represented as transformation rules on that graph. These transformation rules include an original graph that must be matched to a sub graph in the complete state, and a replacing graph that will formally replace the matched sub graph.A graph rewriting system is often made up of a set of pattern broth and replacement graph rewrite rules. In the case of labelled graphs, such as string controlled graphs, a graph rewrite rule is applied to the host graph by looking for an

occurrence of the pattern graph and replacing the detected occurrence with an instance of the graph.

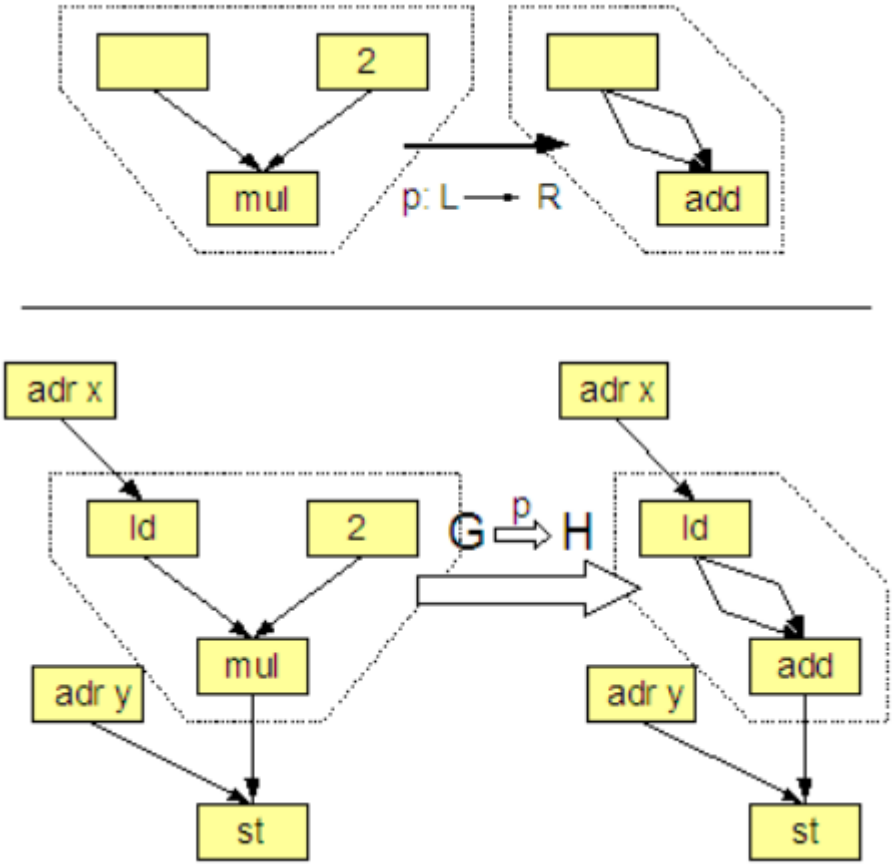


Fig 1.3 :GraphRewriteExample

2)Operation Fusion

Operator fusion is a technique for improving speed by combining one operator (usually an activation function) with another such that they can be executed simultaneously without requiring a memory roundtrip.Many state-of-the-art DNN execution frameworks, such as TensorFlow, TVM, and MNN, use it as a key optimization to improve the efficiency of DNN inference.

Many large-scale machine learning (ML) systems let users specify custom ML algorithms using linear algebra programmes, which are then created automatically.Optimization potential for fused operators—-in terms of fused chains of fundamental operators—-abound in this scenario. These advantages include (1) fewer materialised intermediates, (2) fewer input data scans, and (3) the use of sparsity across operator chains. Automatic operator fusion replaces hand-written fused operators and increases performance for complex or previously unknown chains of processes.Existing fusion heuristics, on the other hand, have a

hard time finding good fusion plans for complicated DAGs or hybrid plans that combine local and distributed activities. We present an optimization framework for systematically reasoning about fusion plans that takes into account materialisation sites in DAGs, sparsity exploitation, distinct fusion template types, and local and distributed operations in this research.We provide algorithms for (1) valid fusion place candidate exploration, (2) cost-based candidate selection, and (3) code creation of local and distributed operations over dense, sparse, and compressed data. Our SystemML studies indicate that optimised fusion plans outperform hand-written fused operators from start to finish, with minimal optimization and code generation overhead.

3)Scheduling

Graph of Tasks Scheduling is the process of transferring a task graph to a destination platform. The application is represented by the task graph: Nodes represent computational jobs, whereas edges provide task precedence constraints. An assignment (the processor that will execute the job) and a schedule (the time when the execution will begin) are assigned to each task. The purpose is to optimise some objective function, most commonly the total execution time, in order to achieve an efficient execution of the application.

1.5) Optimization Types and Methods

We divide optimizations into three categories based on their primary goal: performance, ease of use, and cost savings. The goal of performance optimization is to reduce job execution time and speed up data processing. The purpose is to respond to user requests more quickly.Ease-of-use optimizations are designed to make data processing more user-friendly.

These methods aim to automate the more difficult aspects of data analysis and make application development and deployment easier. Methods for reducing the operational expenses of data analysis systems are referred to as cost reduction optimizations.

--Performance Optimizations. This set of techniques is further divided into two subcategories: (a) single-program optimizations and (b) batch optimizations. Single-program optimizations focus on just one

programme at a time.When used in isolation, these strategies take into account the specifics of a single application. As a result, these strategies ignore interactions with other apps that are running at the same time, as well as possible optimization opportunities. Batch optimizations are strategies that aim to improve the overall performance of a group of applications.These methods look at the system as a place where multiple apps are executing at the same time or for a set amount of time.

-Single-program optimizations. A large number of strategies in this area rely on extracting data from the input and intermediate datasets. These enhancements are aimed at devising more efficient query execution methods. Typically, a user creates a data processing application in a programming language, and the data processing system is in charge of designing an execution strategy for running the application with the resources available. Query planning is the process of converting a user's application into a series of actions that the system can execute.Another type of optimization strategy in this category is involved with accessing data from the processing system's storage efficiently. The purpose is to reduce communication

and data access at random times. Several clever data placement and partitioning algorithms have been developed to decrease data I/O. Predicting which data blocks are likely to be accessed at the same time, for example, and attempting to co-locate them. Efficient data representation, compression strategies, erasure coding, and amortising data skew are all related concerns.

Another type of optimization strategy in this category is involved with accessing data from the processing system's storage efficiently. The purpose is to reduce communication and data access at random times. Several clever data placement and partitioning algorithms have been developed to decrease data I/O. Predicting which data blocks are likely to be accessed at the same time, for example, and attempting to co-locate them. Efficient data representation, compression strategies, erasure coding, and amortisation data skew are all related concerns.When the queries are known ahead of time and the system has enough information about the data distribution, these techniques are very efficient and accurate.

-Batch optimizations. Batch optimization approaches' main purpose is to improve the overall performance of an execution environment, such as a cluster or datacenter. These methods frequently take into account measures such as system throughput and query latency. A large class of batch optimization approaches aims to schedule and manage applications running in the same environment in an efficient manner. Load balancing is another critical issue in distributed setups that has a considerable impact on performance and is strongly tied to scheduling. For simultaneous requests, another frequent option is to use work sharing. This optimization takes a batch of jobs that have been submitted for execution and reorders them to allow

execution pipelines to be shared. Non-concurrent work-sharing is a related strategy that seeks to eliminate unnecessary computations by actualizing and reusing the outcomes of previously run jobs.

-Ease-of-use. The goal of ease-of-use improvements is to make data analysis easier for non-expert users. These strategies mostly consist of tools that analysts can use to increase their efficiency while avoiding poor development practises that can result in problematic products. Parts of the analytic process are automated, while low-level details like parallelization and data distribution are abstracted from the user by tools enabling easy application development. Another set of strategies makes it easier to deploy and maintain analysis software in complex contexts. They might handle fault-tolerance or execute autonomous system configuration and load balancing. Data analysis applications can be tested and debugged with the help of related technologies.

- Cost reduction. The goal of this category of optimizations is to reduce the cost of data processing. Tools for lowering storage requirements, such as data deduplication techniques or erasure coding, are examples of such improvements. Resource and cost-aware scheduling is a large class of cost-cutting optimizations. The system, it is assumed, has information about the costs of various resources and may evaluate various execution techniques and deployments in order to maximise resource utilisation and lower operational expenses. Resource usage optimizations typically improve a system's performance and throughput.

CHAPTER 2:

LITERATURE SURVEY

ONNX Runtime includes a number of graph enhancements that help to increase model performance. Graph optimizations are alterations at the graph level, ranging from minor graph simplifications and node eliminations to more complex node fusions and layout improvements. Graph optimizations are classified into several classes (or levels) based on the quality and functionality they provide. They can be completed on-line or off-line. The optimizations are performed before the inference in online mode, whereas in offline mode, the runtime saves the optimised graph to disc. ONNX Runtime provides Apis for Python, C#, C++, and C to switch between different optimization settings and choose between online and offline.

Basic Graph Improvements These are graph rewrites that keep the semantics while removing superfluous nodes and computation. They are applied to any or all execution providers since they run before graph partitioning. Basic graph optimizations available on the market are as follows: Constant Folding is a method of statically reasoning graph elements that rely exclusively on constant initializers. This reduces the need for them to be computed in real time. Redundant node eliminations: take away all redundant nodes while not dynamic the graph structure. the subsequent such optimizations are presently supported: Identity Elimination Slice Elimination Unsqueeze Elimination Dropout Elimination Semantics-preserving node fusions : Fuse/fold multiple nodes into one node. For example, Conv Add fusion folds the Add operator because the bias of the Conv operator. the subsequent such optimizations are presently supported: Conv Add Fusion Conv Mul Fusion Conv Batch Norm Fusion Relu Clip Fusion Reshape Fusion.

These optimizations involve complex knot fusions. They run after graphics partitioning and only apply to nodes assigned to the CPU or CUDA execution provider. Available advanced graphics optimizations To optimize the inference performance of the BERT model, the approximation in the GELU and Attention Fusion approximation is used for the cuda execution provider. The result may vary slightly. According to our evaluation, the influence on the precision could be neglected: The F1 score for a BERT model in SQuAD v1.1 is almost the same (87.05 vs. 87.03).

The GELU approach is deactivated by default.

These optimizations change the data layout for the corresponding nodes to provide further performance improvements. They run according to plan partitioning and only apply to nodes that are assigned to the CPU execution provider. The available layout optimizations are as follows:

NCHWc Optimizer: Optimizes the graphics by using NCHWc layout instead of NCHW layout.

All optimizations can be done online or offline. When starting an inference session in online mode, we also apply all activated graphics optimizations before starting the model inference. When these

improvements are applied every time we start a session, the model's startup time can increase (particularly for complicated models), which can be critical in production scenarios. The offline mode can bring many advantages here. In offline mode, ONNX Runtime serializes the resulting model on the hard disk after performing graphics optimizations . We may leverage the already optimised model to reduce start-up time later when fresh inference sessions are established for this model.

Levels: The Graph Optimization Level enumeration is defined by the ONNX Runtime to determine which of the above-mentioned optimization levels is active. When you choose a level, you can do optimizations for that level as well as all prior levels. For example, if you activate Extended Optimizations, Basic will also be activated. The mapping of these levels to the enumeration is as follows:

Machine learning mainly evaluates the cost function using labeled samples that are not easy to collect. Semi-supervised learning tries to find a better model based on samples without labels. Most semi-monitored methods are based on a graphical representation of samples and labels (transformed) [11]. For example, the augmentation processes create a new graph that connects original and augmented samples. Graphs, such as data sets with linked samples, were the focus of attention in semi-supervised learning. and so forth.

The goal of any GNN layer is to transform entities while taking into account the structure of the graph by adding information from connected nodes or neighbors.When there is only one graph, the goal of node classification becomes to predict the node labels in a graph while only a subset of the node labels are available (although the model may have access to the properties of the nodes). Further development of Convolutional Neural Networks [14] in Computer Vision [12], Graph Convolutional Network (GCN) [10] uses Laplace graph spectra to filter out signals, and the kernel can be approximated with polynomials or Chebyshev functions [24 , 22]. GCN has become a standard and popular tool in the emerging field of geometric deep learning

From an optimization perspective, Stochastic Gradient Descent (SGD) -based methods using Gradient Estimator have been a popular choice because of their simplicity and efficiency. However, SGD-based algorithms can converge slowly and be difficult to match with large data sets. Adding additional information about gradients can help with convergence, but it is not always possible or easy to obtain. are not easy to calculate, especially at sea level. If the data set is large or the samples are redundant, NNs are trained with methods based on SGD, such as AdaGrad [4] or Adam [9]. These methods use the information from gradients from previous iterations or simply add other parameters such as momentum to the SGD. Natural Gradient Descent (NGD) [1] offers an alternative based on the second moment of gradients.Using an estimate of the inverse of the Fisher information matrix (simply Fisher), NGD

transforms the gradients into so-called natural gradients, which in many cases have been shown to be much faster compared to SGD. The use of NGD enables an efficient investigation of the underlying parameter space geometry in the optimization process. In addition, Fisher information plays a fundamental role in statistical modeling [16]. In frequentist statistics, Fisher information is used to build hypothesis tests and confidence intervals using maximum likelihood estimates. Statistics, defines Jeffreys' Shaper, a commonly used standard shaper for estimating. The methods are heavily dependent on the parameterization. For models with a large number of parameters like DNN, Fisher is so large that it is almost impossible to evaluate natural gradients. Therefore, for a faster calculation, it is preferred to use a Fisher approximation such as KroneckerFactored Approximate Curvature (KFAC) [18], which is easier to store and invert.

The methods are heavily dependent on the parameterization. For models with a large number of parameters like DNN, Fisher is so large that it is almost impossible to evaluate natural gradients. Therefore, for a faster calculation, it is preferred to use a Fisher approximation such as KroneckerFactored Approximate Curvature (KFAC) [18], which is easier to store and invert.

First, graph-based semi-supervised learning with a focus on least squares regression and cross-entropy classification is defined. The necessary background information on optimization and neural networks is provided in the following sections. Let us consider an information source q (x) that generates independent samples xi X, the objective distribution q (yx) assigns yi ∈ Y to every xi and the adjacency distribution q (ax) as q (x) and q (ax, x 0), resulting in a different number of samples from each available distribution.
Let X = X0 be a d0 × n matrix of n ≥ 1

iid xi samples of q (x)

(corresponds to X ∼ q (X)).

Assume n × n adjacency matrix A = [aij] a sample of q (Axi , xj) for i, j = 1, is taken. . . , n (corresponds to A q (AX)).It can be assumed that (X, A) is a graph of n.

Nodes where the i-th column of X shows the covariate at node i and D = diag (P j aij) denotes the diagonal degree matrix. Also denote Y as a dm × n¯ matrix of n and lt;shows yi of q (yxi)

Note that 1 (condition) is 1 if the condition is true, 0 otherwise, 0. Therefore, an empirical cost of can be estimated by

$$\hat{r}(\theta) = 1\ \bar{n}\ X\bar{n}\ i=1\ l(yi\ ,f(xi\ ,\ X,\ A;\ \theta)),$$

The parameter weights are initialized as the original GCN [10] and the input vectors are correspondingly normalized to lines [7]. The model is trained for 200 epochs without an early stop and a learning rate of 0.01. the weight loss of $5 \times 10{-}4$ or the impulse of 0.9.

Optimization performance is measured against both the minimum validation cost and the accuracy of the test to achieve the best validation cost. preconditioned version (AdamKFAC and SGDKFAC). For each method, unmarked samples are used in the training process with a relationship controlled by. 1 shows the costs of the validation of four methods based on Adam (top row) and SGD (bottom row) for the three Citation data sets is shown in Tab.III, IV and V. The test precision values given in the tables are average values of and 95% confidence intervals over 10 runs for the best-fitting hyperparameters in the second division of the CiteSeer data set. The performance of the optimization method as an objective function (cross entropy) is not the same as the prediction function (argmax). In most cases, however, the proposed method will achieve a better accuracy than Adam (the first row in all tables). Since all methods use a fixed learning rate of 0.01, the SGD has a very slow convergence and does not offer competitive results.

The meaning of the hyperparameter γ is shown in Figure 2, Figures 2a and 2d represent the sensitivity of Adam and SGD for the parameter. the faster the convergence. Choosing very small, however, results in larger confidence intervals which are not desirable. The effect of γ on Adam and SGD is shown in Figures 2b and 2e, respectively.Using Adam, due to its faster convergence compared to SGD, a smaller γ, that is, using more predictions, leads to much larger confidence intervals. In other words, the training process dominated by more labels leads to a more stable convergence with a lower one. Therefore, for a stable estimate, λ or γ must be adapted based on their sensitivity to the optimization algorithm.

In this article we present a novel optimization framework for graph-based semi-monitored learning.After the clear definition of semi-monitored problems with the adjacency distribution, we give a comprehensive overview of topics such as semi-monitored learning, the neural network of graph and preconditioning optimization (and NGD as a special case) We used a widely used probabilistic framework that covers least squares regression and cross entropy classification. In the knot classification problem, our proposed method showed an improvement from Adam and SGD not only in the cost of validation but also in accuracy of the GCN test in three divisions of citation records. Extensive experiments on sensitivity to hyperparameters and time complexity were provided.As the first work to our knowledge on preconditioned graphical neural network optimization, we have not only achieved the best test accuracy, but also empirically proven that it can be used with Adam and SGD.

As the preconditioner might considerably have an effect on Adam, illustrating the relation between NGD and Adam and effectively combining them are often a promising direction for future work. we tend to

additionally aim to deploy quicker approximation ways than KFAC like [6] and higher sampling methods for exploiting unlabeled samples. Finally, since this work is especially targeted on single parameter layers, another doable analysis path would be adjusting KFAC to, for example, residual layers

CHAPTER 3:

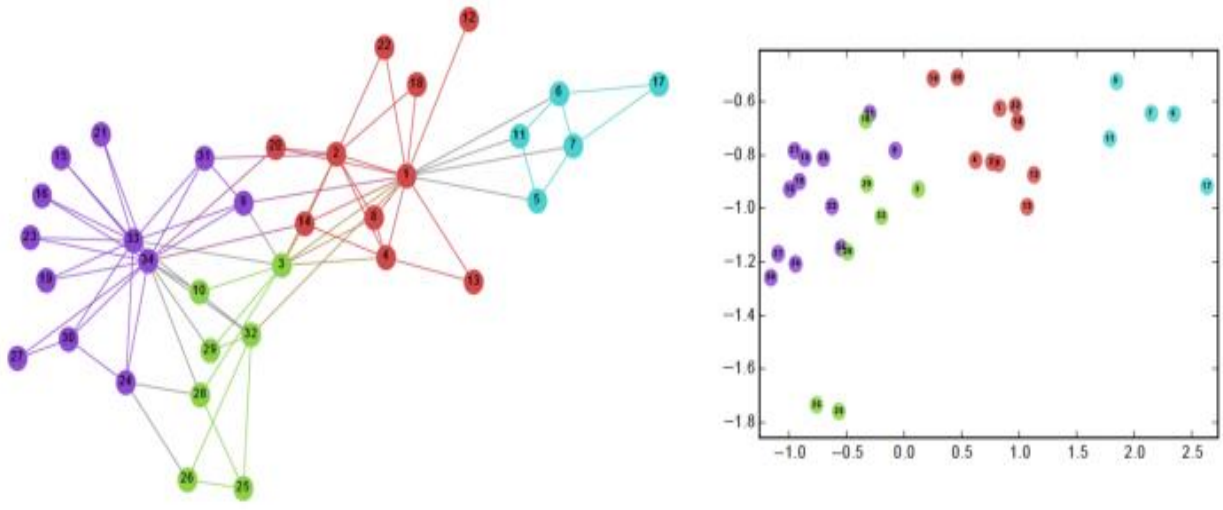
THEORETICAL ANALYSIS

3.1) Algorithms

i) Deepwalk

Because graph data structures may describe complex relationships, new approaches to evaluate and classify entities defined by their interactions have emerged. While these analyses are effective in identifying diverse structures within communities, they lack the ability to store graph features for use in traditional machine learning techniques. [1] Co-interactions inside graphs can be recorded and encoded by basic neural networks into embeddings usable by the aforementioned ML methods, according to DeepWalk's approach.While there are papers that provide easy introductions to the DeepWalk method, there are few that include code and describe implementation specifics for these systems that I could discover. Model parametrization, deployment considerations, and handling unseen data are all covered in detail.

The model has learned a good representation of each node after a DeepWalk GNM has been trained, as seen in the following picture. Distinct shades in the input graph imply different labels. We can see that nodes with the same labels are crowded together in the output graph (embedding with two dimensions), but most nodes with different labels are correctly separated.



(a) Input: Karate Graph (b) Output: Representation

Fig 3.1: example graph of trained deepwalk

DeepWalk is a form of graph neural network [1], which means it works directly on the target graph structure. It makes use of a randomised path traversing technique to reveal localised network topologies. This is accomplished by converting these random pathways into sequences, which are then used to train a Skip-Gram Language Model. To keep things simple in this piece, we'll train our Skip-Gram model using the Gensim module Word2Vec.

Word2vec. The Word2Vec language model is heavily used in this modest version of the DeepWalk algorithm [2]. Word2Vec, introduced by Google in 2013, allowed words to be embedded in n-dimensional space, with similar words clustered together locally. This indicates that cosine distances between words that are frequently used together or in similar situations are less.

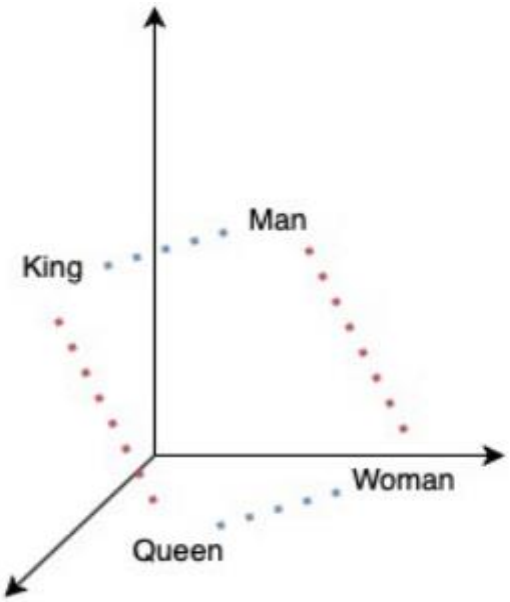


Fig 3.2 : 3- Dim example of word embedding behaviour

Word2Vec does this by comparing the target words to their context using the Skip-Gram technique. At its most basic level, Skip-Gram employs a sliding window technique, in which it attempts to predict the surrounding words based on the target word in the centre. This means that we are effectively trying to estimate the neighbours around the target node inside our network in our use-case of trying to encode comparable nodes within the graph to be close to one other in n-dimensional space.

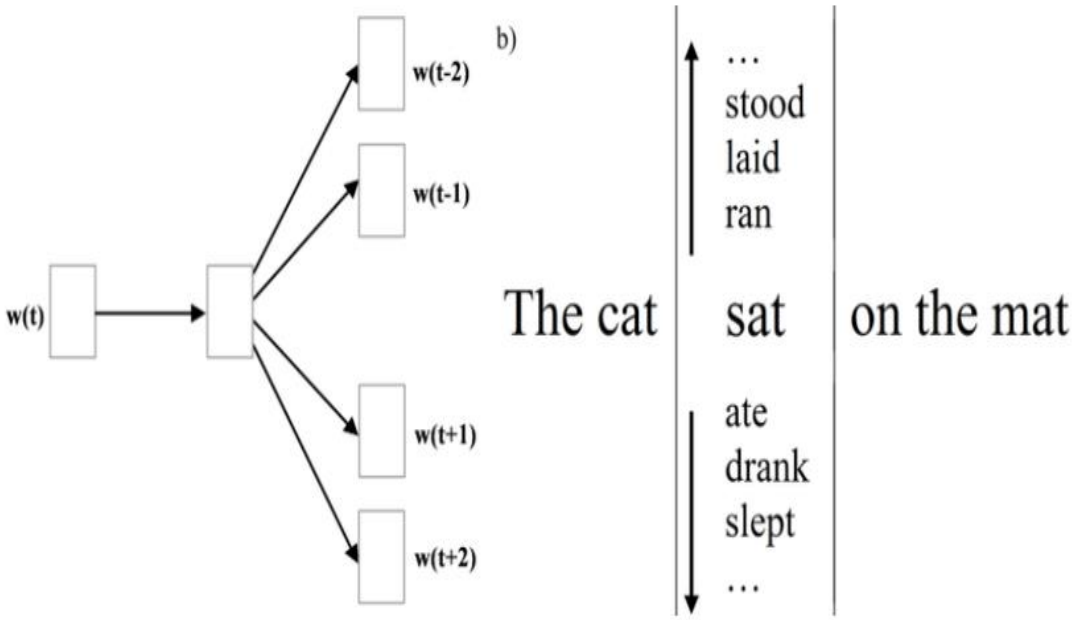


Fig 3.3 : A skim gram example

DeepWalk employs random path-making over graphs to uncover hidden patterns in the network, which are subsequently learned and encoded by neural networks to produce our final embeddings. These random pathways are created in a very straightforward manner: Starting at the target root, randomly select a node's neighbour and add it to the path; then, randomly select another node's neighbour and repeat the process until the necessary number of steps has been taken. This regular sampling of network pathways provides a list of product-ids in the e-commerce case.These IDs are then treated as if they were tokens in a sentence, and a Word2Vec model is used to learn the state-space from them.

To put it another way, the DeepWalk procedure consists of the following steps:

There are a few steps to the DeepWalk process:

1.Perform N "random steps" starting from that node for each node.

2. Each walk should be treated as a series of node-id strings.

3. Train a word2vec model on these string sequences using the Skip-Gram technique, given a list of them.

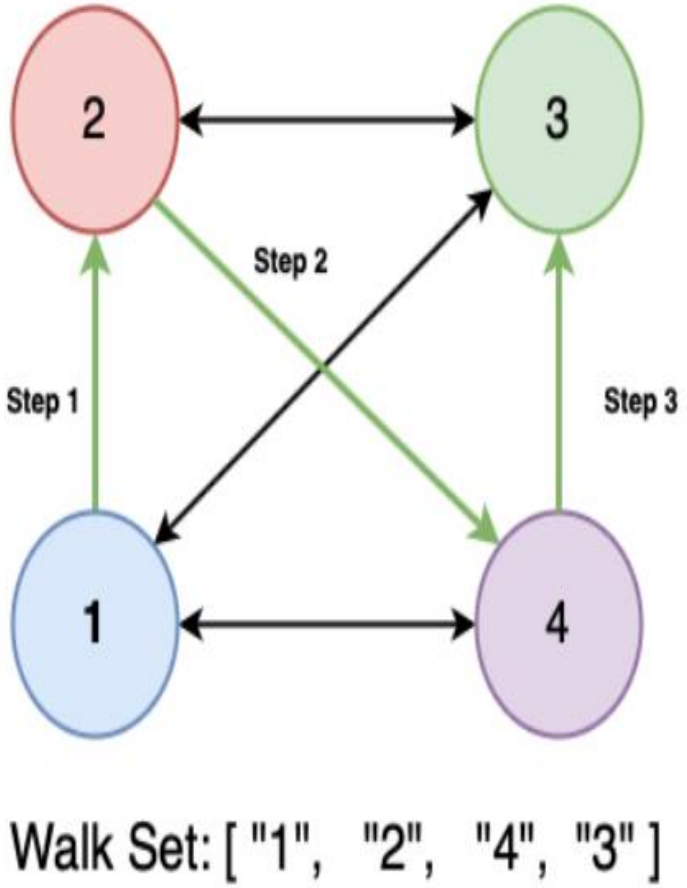


Fig 3.4:walkset example for deep walk

We begin with the network data structure, which defines a set of items identified by their ID. Products that were co-purchased together within the e-commerce ecosystem are referred to as vertices between two products. First, we'll create a function called get random walk(Graph, Node Id):

-Tuning and parameters .Outside of the general model params from our Word2Vec model, there are many features of our DeepWalk that can be parametrized now that we know the basic framework. These may include the following:

→For the W2V training data, the number of random walks was calculated.

→Each walk's depth as measured from the node

On an example dataset, I'll use a general classification strategy to demonstrate how these factors can affect the performance of your model. We're trying to classify the products into their 10 categories in the graph above, which uses a sequence of products and a graph that defines co-purchased products.

Fig 3.5 : walk depth v/s walk count

Using increasing numbers of random walks on the y-axis and increasing random walk depth on the x-axis, the classifier trained on the node vectors from our Word2Vec model improved its classification performance (in accuracy). What we find is that as both parameters increase, accuracy improves, but the

rate of return decreases as they both rise. One thing to keep in mind is that as the number of walks rose, the training time increased linearly, so training times can quickly balloon.

For example, the change in training time from the top left corner to the bottom right corner took only 15 seconds, whereas the difference in training time from the bottom right corner took well over an hour.

ii)Adam

We introduce Adam, a first-order gradient-based stochastic objective function optimization technique. The method is simple to use and is based on adaptive estimates of the gradients' lower-order moments. The method is computationally efficient, requires little memory, and is ideally suited to situations with a lot of data and/or parameters. The method can also be used to solve problems with non-stationary targets and/or very noisy and/or sparse gradients. By adapting to the geometry of the objective function, the approach is invariant to diagonal rescaling of the gradients.

The hyper-parameters have straightforward interpretations and require little adjustment in most cases.

There are some connections to related algorithms that Adam was inspired by. We also look at the algorithm's theoretical convergence features and offer a regret bound on the convergence rate that matches the best known findings in the online convex optimization framework. We show that, when compared to other stochastic optimization approaches, Adam performs well in practise.

We present Adam, a first-order gradient-based optimization technique for stochastic objective functions based on adaptive lower-order moment estimates. The method is simple to develop, computationally efficient, requires minimal memory, is invariant to gradient diagonal rescaling, and is ideally suited for issues with huge amounts of data and/or parameters.The method can also be used to solve problems with non-stationary targets and/or very noisy and/or sparse gradients. The hyper-parameters have straightforward interpretations and require little adjustment in most cases. There are some connections to related algorithms that Adam was inspired by. We also look at the algorithm's theoretical convergence features and offer a regret bound on the convergence rate that matches the best known findings in the online convex optimization framework. Adam performs well in practise and compares favourably to other stochastic optimization approaches, according to empirical evidence.

iii)GraphSAGE

The inductive algorithm GraphSAGE is used to compute node embeddings. GraphSAGE generates node embeddings on unseen nodes or graphs using node feature information. Rather than learning distinct embeddings for each node, the algorithm learns a function that builds embeddings by sampling and aggregating data from a node's immediate surroundings.

Hamilton, Ying, and Leskovec devised the GraphSage algorithm, which is an inductive deep learning method for graphs that generates low-dimensional vector representations for nodes. This differs from prior graph machine learning approaches such as Graph Convolutional Networks or DeepWalk, which are fundamentally transductive, meaning they can only produce embeddings for nodes existing in the fixed graph during training.This means that if the graph evolves in the future and new nodes (not observed during training) enter the graph, we will need to retrain the entire graph in order to compute the embeddings for the new node.

Because of their inability to generalise on unseen nodes, transductive techniques are inefficient when applied to ever-evolving graphs (such as social networks, protein-protein networks, and so on). Transductive techniques also have the drawback of not being able to use node properties such as text attributes, node profile information, node degrees, and so on.

The GraphSage approach, on the other hand, uses both the rich node attributes and the topological structure of each node's neighbourhood to produce representations for new nodes efficiently and without retraining.

GraphSage offers a solution to the aforementioned challenge by inductively learning the embedding for each node. Each node is represented by the aggregation of its immediate surroundings. As a result, even if a new node appears in the graph that was not present during training, its neighbours can still appropriately represent it.

Working principles of GraphSage :

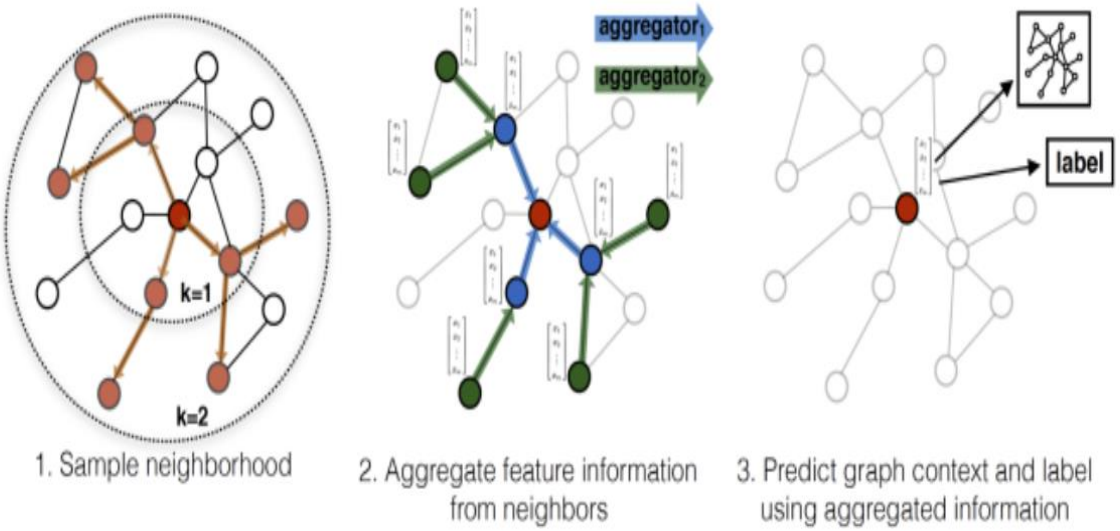


Fig 3.6 : working principles of graphsage

GraphSage's operation is broken into two steps: the first is neighbourhood sampling of an input graph, and the second is learning aggregation functions at each search depth. We'll go over each of these processes in depth, beginning with a brief explanation of why sampling nodes in the area was necessary. Following that, we'll go through the importance of learning aggregator functions, which essentially let the GraphSage algorithm attain its inductiveness property.

Its inductive property is based on the assumptions that we don't need to learn the embeddings for each node, but rather learn an aggregation function (could be any differentiable function like mean, pooling, or lstm) that knows how to aggregate information (or features) from a node's local neighbourhood (learning takes place via stochastic gradient descent) such that the aggregated feature representation of a node v now includes the information (or features).

CHAPTER 4:

EXPERIMENTAL INVESTIGATION

4.1) Graph Optimization in Onnx

While loading a transformer model, ONNX Runtime performs most optimizations automatically. This tool, which tweaks models for the optimal performance, contains some of the most recent optimizations that have not yet been implemented into ONNX Runtime.

The following scenarios can benefit from this tool:

- Tf2onnx or keras2onnx export the model, and ONNX Runtime does not currently support graph optimization for them.
- Convert the model to float16 to improve performance on GPUs with Tensor Cores when utilising mixed precision (like V100 or T4).
- Due to form inference, the model has inputs with dynamic axes, which prevents several optimizations from being applied in ONNX Runtime.
- To see how some fusions affect performance or accuracy, disable or enable them.

Limitations - The maximum number of attention heads is 1024 due to the CUDA implementation of the Attention kernel. The maximum permitted sequence length for Longformer is 4096 bytes, and 1024 bytes for other types of models.

- Example : Scikit-learn Logistic regression
  - Step 1 : Train a model , we use iris data set

```
In [1]: from sklearn.datasets import load_iris
        from sklearn.model_selection import train_test_split
        iris = load_iris()
        X, y = iris.data, iris.target
        X_train, X_test, y_train, y_test = train_test_split(X, y)

        from sklearn.linear_model import LogisticRegression
        clr = LogisticRegression()
        clr.fit(X_train, y_train)
        print(clr)
```

Fig 4.1.1: code snippet 1

- Step 2: Convert or export the model into onnx format

```
In [3]: from skl2onnx import convert_sklearn
        from skl2onnx.common.data_types import FloatTensorType

        initial_type = [('float_input', FloatTensorType([None, 4]))]
        onx = convert_sklearn(clr, initial_types=initial_type)
        with open("logreg_iris.onnx", "wb") as f:
            f.write(onx.SerializeToString())
```

Fig 4.1.2: code snippet 2

- Step 3: Load and run the model using onnx runtime

```
In [4]: import numpy
        import onnxruntime as rt

        sess = rt.InferenceSession("logreg_iris.onnx")
        input_name = sess.get_inputs()[0].name
        pred_onx = sess.run(None, {input_name: X_test.astype(numpy.float32)})[0]
        print(pred_onx)
```

Fig 4.1.3: code snippet 3

The output for the above code displayed in the *Fig 4.3: code snippet 3* is as follows:

```
[0 0 1 0 2 0 2 2 2 1 1 2 2 1 0 0 2 2 1 0 0 0 2 1 2 2 2 1 0 1 0 1 1 0 0 0 2
 2]
```

Fig 4.1.4 : output

4.2) Using a pre-trained onnx model for interfacing

- We'll need Python 3.x installed on your PC to do this. To begin, we'll create a Python3 virtual environment to separate it from the machine's primary Python environment.
- Let's set up the virtual environment and install the Python modules we'll require for our programme. In your environment, run the following command to install ONNX, ONNX Runtime, and OpenCV.
- Now from the ONNX Model Zoo, download and expand the MNIST pre-trained model that was trained in Microsoft CNTK Toolkit.

→python3 -m venvonnx_mnist

→sourceonnx_mnist/bin/activate

→pip installonnxonnxruntimeopencv-python

→wget https://www.cntk.ai/OnnxModels/mnist/opset_7/mnist.tar.gz

→tarxvcf mnist.tar.gz

- Now we're taken to a new directory named minst, which contains the model and test data serialised as ProtoBuf files.
- As demonstrated in the graph from Netron, the MNIST model from the ONNX Model Zoo uses maxpooling to update the weights in its convolutions.
- Two convolutional layers, two maxpool layers, one dense layer, and an output layer can categorise one of the 10 labels used in the MNIST dataset.

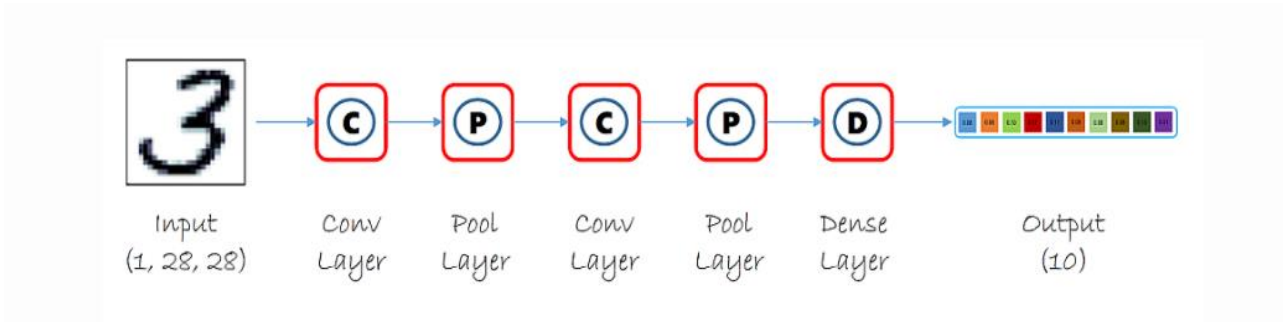


Fig 4.2.1 :minst layers

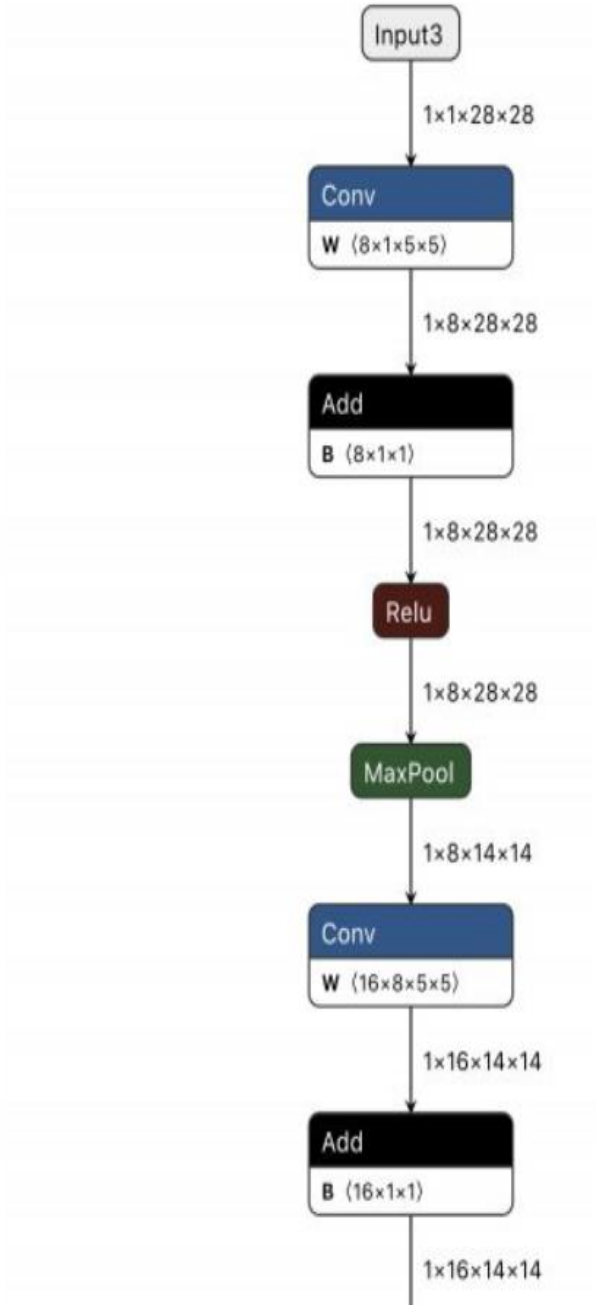


Fig 4.2.2 :minst model

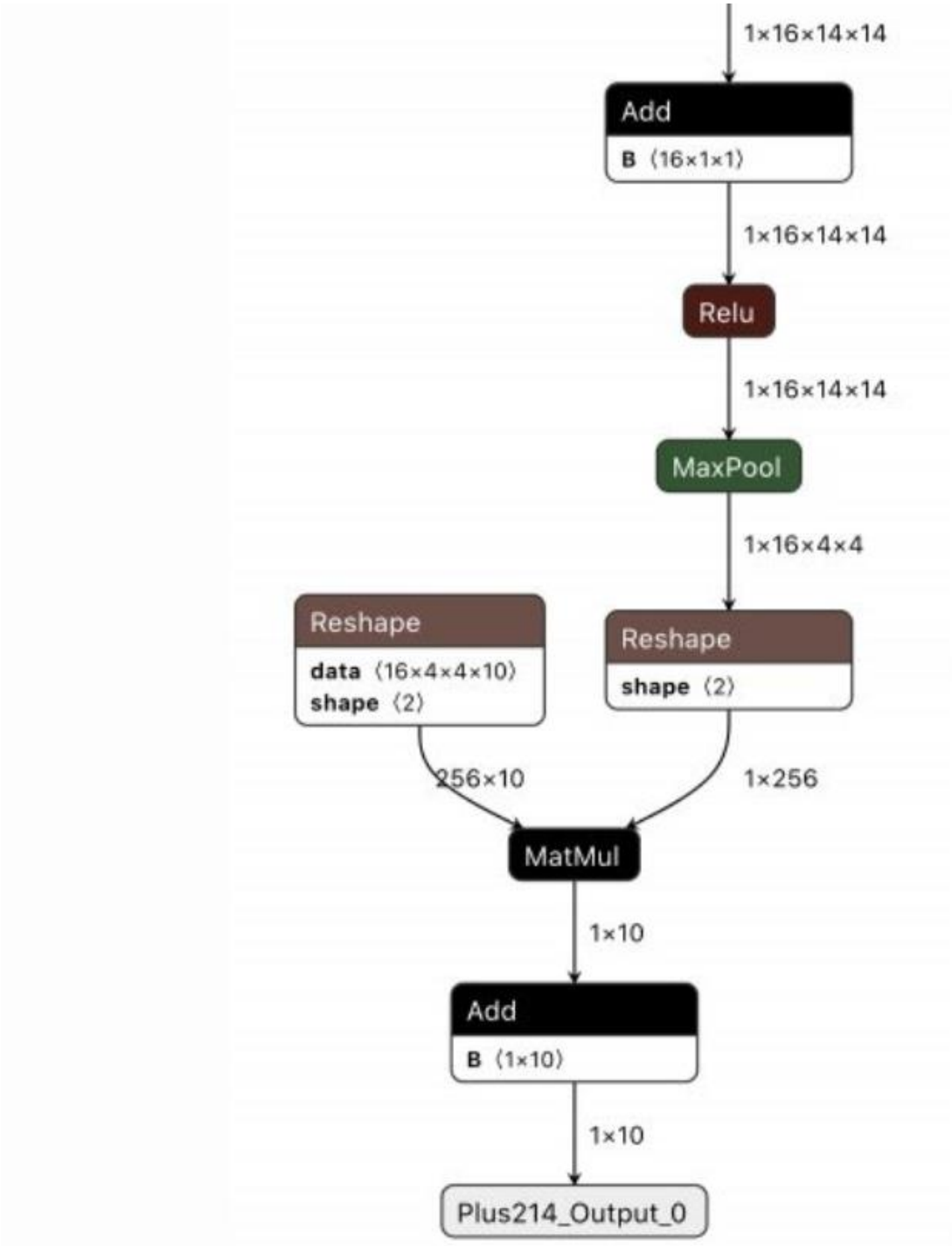


Fig 4.2.3 :minst modelcont

The both above figures *Fig 4.2.2 :minst model*and*Fig 4.2.3: minstmodelcont*together is the minst model

- We'll now develop code to do inference on the MNIST model that has already been trained.
- OnnX, ONNX Runtime, and the NumPy auxiliary modules related to ONNX are all used.
- The ONNX module aids in the parsing of the model file, whereas the ONNX Runtime module handles session creation and inference.
- After that, we'll set up some variables to store the model files' paths and command-line options.
- The image will be loaded and preprocessed with OpenCV in the next phase.
- The image will now be converted to a NumPy array of type float32.
- The input layer and output layer of the neural network must have the same name. The session.getinputs() and session.getoutputs() methods make it simple to get them. The output from the above sample matches the names of the input and output nodes displayed by Netron.
- To find the value with the highest probability, we use NumPy'sargmax function.

- The code is as follows :

```
importjson
import sys
importos
import time
importnumpy as np
import cv2
importonnx
importonnxruntime
fromonnx import numpy_helper

model_dir ="./mnist"
model=model_dir+"/model.onnx"
path=sys.argv[1]
```

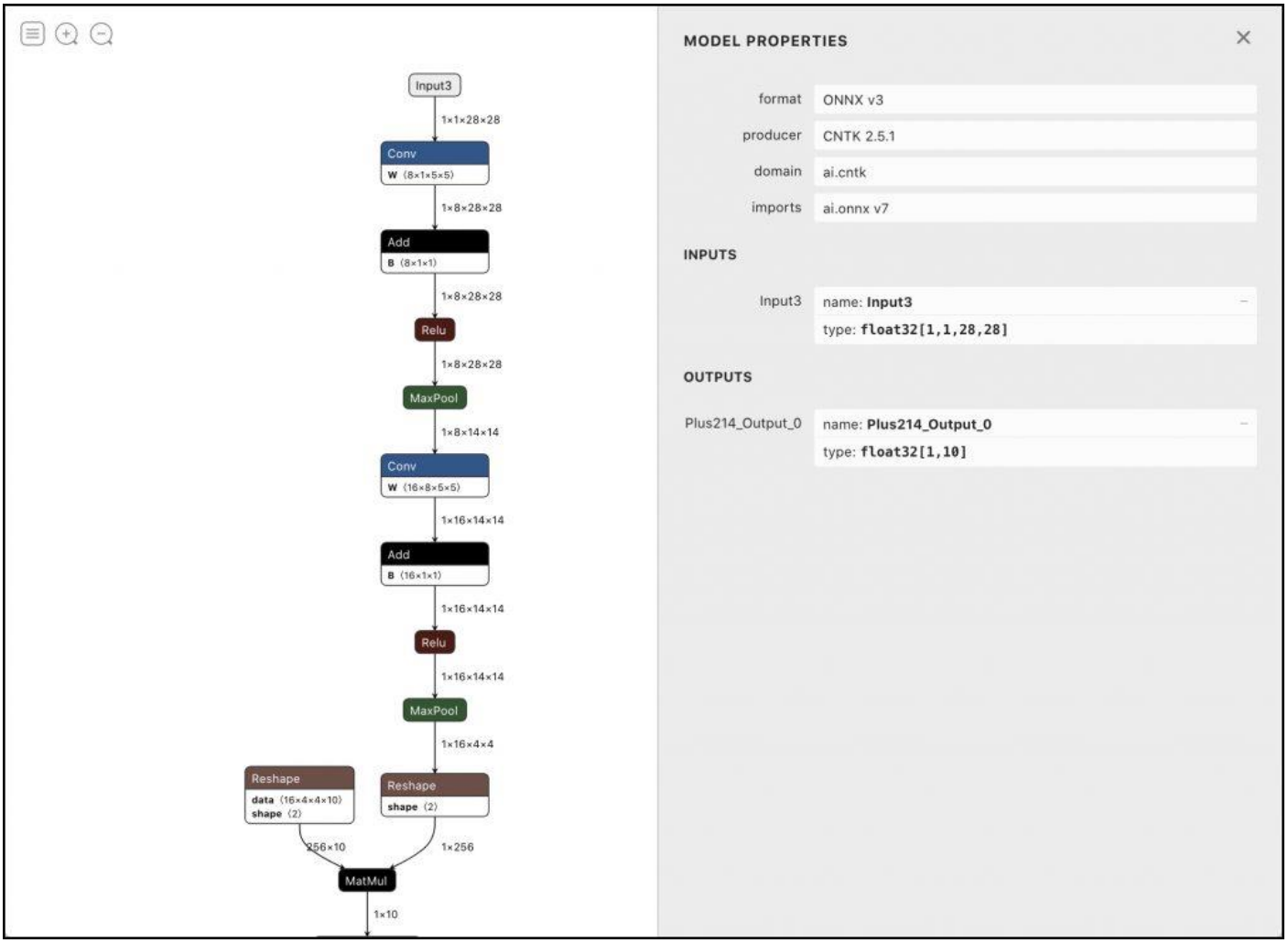
```
#Preprocess the image
img = cv2.imread(path)
img = np.dot(img[...,:3], [0.299, 0.587, 0.114])
img = cv2.resize(img, dsize=(28, 28), interpolation=cv2.INTER_AREA)
img.resize((1, 1, 28, 28))

data = json.dumps({'data': img.tolist()})



data = np.array(json.loads(data)['data']).astype('float32')
session = onnxruntime.InferenceSession(model, None)
input_name = session.get_inputs()[0].name
output_name = session.get_outputs()[0].name

result = session.run([output_name], {input_name: data})
prediction=int(np.argmax(np.array(result).squeeze(), axis=0))
print(prediction)
```


Input3
1×1×28×28
Conv
W (8×1×5×5)
1×8×28×28
Add
B (8×1×1)
1×8×28×28
Relu
1×8×28×28
MaxPool
1×8×14×14
Conv
W (16×8×5×5)
1×16×14×14
Add
B (16×1×1)
1×16×14×14
Relu
1×16×14×14
MaxPool
1×16×4×4
Reshape
data (16×4×4×10)
shape (2)
Reshape
shape (2)
256×10
1×256
MatMul
1×10
MODEL PROPERTIES
format ONNX v3
producer CNTK 2.5.1
domain ai.cntk
imports ai.onnx v7
INPUTS
Input3 name: Input3
type: float32[1,1,28,28]
OUTPUTS
Plus214_Output_0 name: Plus214_Output_0
type: float32[1,10]

CHAPTER 5:
EXPERIMENTAL ANALYSIS

```
[ ] pip install numpy

Requirement already satisfied: numpy in /usr/local/lib/python3.7/dist-packages (1.19.5)
```

```
pip install mxnet

Collecting mxnet
  Downloading mxnet-1.8.0.post0-py2.py3-none-manylinux2014_x86_64.whl (46.9 MB)
     |████████████████████████████████| 46.9 MB 47 kB/s
Requirement already satisfied: requests<3,>=2.20.0 in /usr/local/lib/python3.7/dist-packages (from mxnet) (2.23.0)
Requirement already satisfied: numpy<2.0.0,>1.16.0 in /usr/local/lib/python3.7/dist-packages (from mxnet) (1.19.5)
Requirement already satisfied: graphviz<0.9.0,>=0.8.1 in /usr/local/lib/python3.7/dist-packages (from mxnet) (0.8.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.20.0->mxnet) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.20.0->mxnet) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.20.0->mxnet) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests<3,>=2.20.0->mxnet) (2021.10.8)
Installing collected packages: mxnet
Successfully installed mxnet-1.8.0.post0
```

```
import matplotlib
import mxnet as mx
import numpy as np
from mxnet import gluon, nd
from mxnet import autograd as ag
from mxnet.gluon import nn
from mxnet.gluon.data.vision import transforms

from gluoncv.data import imagenet
from gluoncv.utils import makedirs, TrainingHistory
```

```
[ ] import os
from mxnet.context import cpu
from mxnet.gluon.block import HybridBlock
from mxnet.gluon.contrib.nn import HybridConcurrent
import multiprocessing

/usr/local/lib/python3.7/dist-packages/gluoncv/__init__.py:40: UserWarning: Both `mxnet==1.8.0` and `torch==1.10.0+cu111` are installed. You might encounter inc
  warnings.warn(f'Both `mxnet=={mx.__version__}` and `torch=={torch.__version__}` are installed. '
```

```
model_name = 'mobilenetv2_1.0'

# path to training and validation images to use
data_dir = '/home/ubuntu/imagenet/img_dataset'

# training batch size per device (CPU/GPU)
batch_size = 40

# number of GPUs to use (automatically detect the number of GPUs)
num_gpus = len(mx.test_utils.list_gpus())

num_workers = multiprocessing.cpu_count()
num_epochs = 1
lr = 0.045
momentum = 0.9

wd = 0.00004
```

```
]
lr_decay_period = 1

lr_decay_epoch = '30,60,90'
mode = 'hybrid'
log_interval = 50
save_frequency = 10
save_dir = 'params'
logging_dir = 'logs'
save_plot_dir = '.'
```

```
class RELU6(nn.HybridBlock):
    """Relu6 used in MobileNetV2."""

    def __init__(self, **kwargs):
        super(RELU6, self).__init__(**kwargs)

    def hybrid_forward(self, F, x):
        return F.clip(x, 0, 6, name="relu6")


def _add_conv(out, channels=1, kernel=1, stride=1, pad=0,
              num_group=1, active=True, relu6=False):
    out.add(nn.Conv2D(channels, kernel, stride, pad, groups=num_group, use_bias=False))
    out.add(nn.BatchNorm(scale=True))
    if active:
        out.add(RELU6() if relu6 else nn.Activation('relu'))
```

```
def _add_conv_dw(out, dw_channels, channels, stride, relu6=False):
    _add_conv(out, channels=dw_channels, kernel=3, stride=stride,
              pad=1, num_group=dw_channels, relu6=relu6)
    _add_conv(out, channels=channels, relu6=relu6)


class LinearBottleneck(nn.HybridBlock):
    r"""LinearBottleneck used in MobileNetV2 model from the
    `"Inverted Residuals and Linear Bottlenecks:
      Mobile Networks for Classification, Detection and Segmentation"
    <https://arxiv.org/abs/1801.04381>`_ paper.
    Parameters
    ----------
    in_channels : int
        Number of input channels.
    channels : int
        Number of output channels.
    t : int
        Layer expansion ratio.
    stride : int
        stride
    """

    def __init__(self, in_channels, channels, t, stride, **kwargs):
        super(LinearBottleneck, self).__init__(**kwargs)
        self.use_shortcut = stride == 1 and in_channels == channels
        with self.name_scope():
            self.out = nn.HybridSequential()
```

```
            _add_conv(self.out, in_channels * t, relu6=True)
            _add_conv(self.out, in_channels * t, kernel=3, stride=stride,
                      pad=1, num_group=in_channels * t, relu6=True)
            _add_conv(self.out, channels, active=False, relu6=True)

    def hybrid_forward(self, F, x):
        out = self.out(x)
        if self.use_shortcut:
            out = F.elemwise_add(out, x)
        return out


# Net
class MobileNetV2(nn.HybridBlock):
    r"""MobileNetV2 model from the
    `"Inverted Residuals and Linear Bottlenecks:
      Mobile Networks for Classification, Detection and Segmentation"
    <https://arxiv.org/abs/1801.04381>`_ paper.
    Parameters
    ----------
    multiplier : float, default 1.0
        The width multiplier for controling the model size. The actual number of channels
        is equal to the original channel size multiplied by this multiplier.
    classes : int, default 1000
        Number of classes for the output layer.
    """

    def __init__(self, multiplier=1.0, classes=1000, **kwargs):
```

```
    def __init__(self, multiplier=1.0, classes=1000, **kwargs):
        super(MobileNetV2, self).__init__(**kwargs)
        with self.name_scope():
            self.features = nn.HybridSequential(prefix='features_')
            with self.features.name_scope():
                _add_conv(self.features, int(32 * multiplier), kernel=3,
                          stride=2, pad=1, relu6=True)

                in_channels_group = [int(x * multiplier) for x in [32] + [16] + [24] * 2
                                     + [32] * 3 + [64] * 4 + [96] * 3 + [160] * 3]
                channels_group = [int(x * multiplier) for x in [16] + [24] * 2 + [32] * 3
                                  + [64] * 4 + [96] * 3 + [160] * 3 + [320]]
                ts = [1] + [6] * 16
                strides = [1, 2] * 2 + [1, 1, 2] + [1] * 6 + [2] + [1] * 3

                for in_c, c, t, s in zip(in_channels_group, channels_group, ts, strides):
                    self.features.add(LinearBottleneck(in_channels=in_c, channels=c,
                                                       t=t, stride=s))

                last_channels = int(1280 * multiplier) if multiplier > 1.0 else 1280
                _add_conv(self.features, last_channels, relu6=True)

                self.features.add(nn.GlobalAvgPool2D())

            self.output = nn.HybridSequential(prefix='output_')
            with self.output.name_scope():
                self.output.add(
                    nn.Conv2D(classes, 1, use_bias=False, prefix='pred_'),
                    nn.Flatten()
```

```
    def hybrid_forward(self, F, x):
        x = self.features(x)
        x = self.output(x)
        return x


# Constructor
def get_mobilenet_v2(multiplier, **kwargs):
    r"""MobileNetV2 model from the
    `"Inverted Residuals and Linear Bottlenecks:
      Mobile Networks for Classification, Detection and Segmentation"
    <https://arxiv.org/abs/1801.04381>`_ paper.
    Parameters
    ----------
    multiplier : float
        The width multiplier for controling the model size. Only multipliers that are no
        less than 0.25 are supported. The actual number of channels is equal to the original
        channel size multiplied by this multiplier.
    """
    net = MobileNetV2(multiplier, **kwargs)
    return net

def mobilenet_v2_1_0(**kwargs):
    r"""MobileNetV2 model from the
    `"Inverted Residuals and Linear Bottlenecks:
      Mobile Networks for Classification, Detection and Segmentation"
    <https://arxiv.org/abs/1801.04381>`_ paper.
    """
    return get_mobilenet_v2(1.0, **kwargs)
```

```
logging.basicConfig(level=logging.INFO)

# Specify classes (1000 for ImageNet)
classes = 1000
# Extrapolate batches to all devices
batch_size *= max(1, num_gpus)
# Define context
context = [mx.gpu(i) for i in range(num_gpus)] if num_gpus > 0 else [mx.cpu()]

lr_decay_epoch = [int(i) for i in lr_decay_epoch.split(',')] + [np.inf]

kwargs = {'classes': classes}

# Define optimizer (nag = Nestrov Accelerated Gradient)
optimizer = 'nag'
optimizer_params = {'learning_rate': lr, 'wd': wd, 'momentum': momentum}

# Retireve gluon model
net = models[model_name](**kwargs)

# Define accuracy measures - top1 error and top5 error
acc_top1 = mx.metric.Accuracy()
acc_top5 = mx.metric.TopKAccuracy(5)
train_history = TrainingHistory(['training-top1-err', 'training-top5-err',
                                 'validation-top1-err', 'validation-top5-err'])
makedirs(save_dir)
```

```
ormalize = transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
jitter_param = 0.0
lighting_param = 0.0

# Input pre-processing for train data
def preprocess_train_data(normalize, jitter_param, lighting_param):
    transform_train = transforms.Compose([
        transforms.Resize(480),
        transforms.RandomResizedCrop(224),
        transforms.RandomFlipLeftRight(),
        transforms.RandomColorJitter(brightness=jitter_param, contrast=jitter_param,
                                     saturation=jitter_param),
        transforms.RandomLighting(lighting_param),
        transforms.ToTensor(),
        normalize
    ])
    return transform_train

# Input pre-processing for validation data
def preprocess_test_data(normalize):
    transform_test = transforms.Compose([
        transforms.Resize(256),
        transforms.CenterCrop(224),
        transforms.ToTensor(),
        normalize
    ])
    return transform_test
```

```
[ ] def test(ctx, val_data):
        # Reset accuracy metrics
        acc_top1.reset()
        acc_top5.reset()
        for i, batch in enumerate(val_data):
            # Load validation batch
            data = gluon.utils.split_and_load(batch[0], ctx_list=ctx, batch_axis=0)
            label = gluon.utils.split_and_load(batch[1], ctx_list=ctx, batch_axis=0)
            # Perform forward pass
            outputs = [net(X) for X in data]
            # Update accuracy metrics
            acc_top1.update(label, outputs)
            acc_top5.update(label, outputs)
        # Retrieve and return top1 and top5 errors
        _, top1 = acc_top1.get()
        _, top5 = acc_top5.get()
        return (1-top1, 1-top5)
```

```
def train(epochs, ctx):
    if isinstance(ctx, mx.Context):
        ctx = [ctx]
    # Initialize network - Use method in MSRA paper <https://arxiv.org/abs/1502.01852>
    net.initialize(mx.init.MSRAPrelu(), ctx=ctx)
    # Prepare train and validation batches
    transform_train = preprocess_train_data(normalize, jitter_param, lighting_param)
    transform_test = preprocess_test_data(normalize)
    train_data = gluon.data.DataLoader(
        imagenet.classification.ImageNet(data_dir, train=True).transform_first(transform_train),
        batch_size,batch_size, shuffle,True, last_batch,'discard', num_workers,num_workers)
```

```
lr_decay_count = 0

best_val_score = 1
# Main training loop - loop over epochs
for epoch in range(epochs):
    tic = time.time()
    # Reset accuracy metrics
    acc_top1.reset()
    acc_top5.reset()
    btic = time.time()
    train_loss = 0
    num_batch = len(train_data)

    # Check and perform learning rate decay
    if lr_decay_period and epoch and epoch % lr_decay_period == 0:
        trainer.set_learning_rate(trainer.learning_rate*lr_decay)
    elif lr_decay_period == 0 and epoch == lr_decay_epoch[lr_decay_count]:
        trainer.set_learning_rate(trainer.learning_rate*lr_decay)
        lr_decay_count += 1
    # Loop over batches in an epoch
    for i, batch in enumerate(train_data):
        # Load train batch
        data = gluon.utils.split_and_load(batch[0], ctx_list=ctx, batch_axis=0)
        label = gluon.utils.split_and_load(batch[1], ctx_list=ctx, batch_axis=0)
        label_smooth = label
        # Perform forward pass
        with ag.record():
            outputs = [net(X) for X in data]
            loss = [L(yhat, y) for yhat, y in zip(outputs, label_smooth)]
        # Perform backward pass
```

```
        # Retrieve training errors and loss
        _, top1 = acc_top1.get()
        _, top5 = acc_top5.get()
        err_top1, err_top5 = (1-top1, 1-top5)
        train_loss /= num_batch * batch_size

        # Compute validation errors
        err_top1_val, err_top5_val = test(ctx, val_data)
        # Update training history
        train_history.update([err_top1, err_top5, err_top1_val, err_top5_val])
        # Update plot
        train_history.plot(['training-top1-err', 'validation-top1-err','training-top5-err', 'validation-top5-err'],
                           save_path='%s/%s_top_error.png'%(save_plot_dir, model_name))

        # Log training progress (after each epoch)
        logging.info('[Epoch %d] training: err-top1=%f err-top5=%f loss=%f'%(epoch, err_top1, err_top5, train_loss))
        logging.info('[Epoch %d] time cost: %f'%(epoch, time.time()-tic))
        logging.info('[Epoch %d] validation: err-top1=%f err-top5=%f'%(epoch, err_top1_val, err_top5_val))

        # Save a snapshot of the best model - use net.export to get MXNet symbols and params
        if err_top1_val < best_val_score and epoch > 50:
            best_val_score = err_top1_val
            net.export('%s/%.4f-imagenet-%s-best'%(save_dir, best_val_score, model_name), epoch)
        # Save a snapshot of the model after each 'save_frequency' epochs
        if save_frequency and save_dir and (epoch + 1) % save_frequency == 0:
            net.export('%s/%.4f-imagenet-%s'%(save_dir, best_val_score, model_name), epoch)
    # Save a snapshot of the model at the end of training
    if save_frequency and save_dir:
        net.export('%s/%.4f-imagenet-%s'%(save_dir, best_val_score, model_name), epochs-1)
```

```
[ ] def main():
        net.hybridize()
        train(num_epochs, context)
```

```
[ ] import mxnet as mx
    import numpy as np
    from mxnet.contrib import onnx as onnx_mxnet
```

```
[ ]
```

```
[ ] path='http://data.mxnet.io/models/imagenet/'
    [mx.test_utils.download(path+'resnet/18-layers/resnet-18-0000.params'),
     mx.test_utils.download(path+'resnet/18-layers/resnet-18-symbol.json'),
     mx.test_utils.download(path+'synset.txt')]
```

```
['resnet-18-0000.params', 'resnet-18-symbol.json', 'synset.txt']
```

```
help(onnx_mxnet.export_model)
```

```
Help on function export_model in module mxnet.contrib.onnx.mx2onnx.export_model:

export_model(sym, params, input_shape, input_type=<class 'numpy.float32'>, onnx_file_path='model.onnx', verbose=False, opset_version=None)
    Exports the MXNet model file, passed as a parameter, into ONNX model.
```

```
[ ]     onnx_file_path : str
            Onnx file path

        Notes
        -----
        This method is available when you ``import mxnet.contrib.onnx``
```

```
[ ] sym = 'resnet-18-symbol.json'
    params = 'resnet-18-0000.params'
    # Standard Imagenet input - 3 channels, 224*224
    input_shape = (1,3,224,224)
    # Path of the output file
    onnx_file = 'mxnet_exported_resnet50.onnx'
```

```
pip install onnx
```

```
Collecting onnx
  Downloading onnx-1.10.2-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (12.7 MB)
     |████████████████████████████████| 12.7 MB 83 kB/s
Requirement already satisfied: numpy>=1.16.6 in /usr/local/lib/python3.7/dist-packages (from onnx) (1.19.5)
Requirement already satisfied: protobuf in /usr/local/lib/python3.7/dist-packages (from onnx) (3.17.3)
Requirement already satisfied: typing-extensions>=3.6.2.1 in /usr/local/lib/python3.7/dist-packages (from onnx) (3.10.0.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages (from onnx) (1.15.0)
Installing collected packages: onnx
Successfully installed onnx-1.10.2
```

```
[ ]
```

CHAPTER 6:
DISCUSSION OF RESULTS

MobileNetV2 could be a convolution neural detail that looks to perform well on cell phones. it's upheld Associate in Nursing modified remaining design any place the lingering associations are between the bottleneck layers. The halfway development layer utilizes light-weight depthwise convolutions to channel choices as a stock of non-linearity. Overall, the plan of MobileNetV2 contains the underlying absolutely convolution layer with 32 channels, trailed by nineteen leftover bottleneck layers. In this mode we have changed over a model into onnx stage and we observed a few advancements.

CHAPTER 7:

CONCLUSION

Debugging the Optimization solution is more challenging than debugging the Rulebased Simulation solution. Because it is a global solution, there is rarely a specific explanation for a poor output or an unusual result. It could be a mix of numerous limitations and input data from multiple places and timesteps. In the presence of identified limitations, constrained optimization is a set of approaches for identifying the best solution (the optimal solution) to a problem defined by a number of viable solutions. Traditional optimization techniques can be used to identify the best solution or unconstrained maxima and minima of continuous and differentiable functions. These are mathematical methods that use differential calculus to get the best solution. Problems in which a function must be reduced or maximised while adhering to constraints are known as constrained optimization problems. In various fields of study, optimization methods are used to identify solutions that maximise or minimise specific study parameters, such as minimising expenses in the manufacture of a thing or service, maximising earnings, minimising raw material in the development of a good, or maximising productivity. Discrete optimization requires some or all of the variables in a model to belong to a discrete set, as opposed to continuous optimization, which allows the variables to take on any value within a range of values. Maximizing or minimising a function in relation to a set, which frequently represents a range of options in a given circumstance. The tool allows you to compare different options to see which one is the "ideal." The first of them takes the hardness of all feasible algorithms for the optimization problem at hand and averages it. We show that there is no difference amongst optimization issues based on this amount, and that no problem is intrinsically tougher than others. As a result, optimization methods like stochastic gradient descent, min-batch gradient descent, gradient descent with momentum, and the Adam optimizer are extremely important. Our neural network can learn thanks to these techniques. In terms of speed, however, certain strategies outperform others. When fitting a machine learning algorithm, function optimization is the reason for minimising error, cost, or loss. In a predictive modelling project, optimization is also done during data preparation, hyperparameter tweaking, and model selection. The necessity for an interface that facilitates inference in various hardware architectures led to the creation of ONNX Runtime. It was exceedingly expensive to deploy models that were designed primarily for CUDA-based architectures to NUPHAR, nGraph, OpenVINO-based architectures, and so on before ONNX Runtime. In other words, the framework and the hardware architecture for which the model was designed were both interdependent. With the ONNX standard and the ONNX Runtime accelerator, a wide range of interoperability between frameworks and hardware architectures is now possible. Inference time is significantly lowered due to improved inference performance. Training time has been cut down.Models can be created and trained in Python and then deployed in C, C++, or Java applications. It gives data scientists complete freedom to train a model in one framework and generate inference in another. Hardware Optimizations - Using Open Neural Network

Exchange, it's simple to provide Data Scientists the capacity to optimise. Even in this scenario, ONNX inferences/predictions are 6–7 times faster than TensorFlow's initial model. As previously said, if you deal with larger datasets, the outcomes will be considerably more striking. The optimum setup for the T4 is to run ONNX with batches of 8 samples, which results in a 12x speedup over batch size 1 on pytorch. We can obtain up to a 28x speedup on the V100 with batches of 32 or 64 compared to the baseline for GPU and 90x for the baseline on CPU. ONNX Runtime also includes a mixed precision version that allows more training data to be stored in the accessible memory of a single NVIDIA GPU, allowing training processes to converge faster and save time. It's integrated into PyTorch and TensorFlow's existing trainer code. ONNX Runtime includes considerable production-level optimization, testing, and other enhancements.

CHAPTER 8:

REFERENCES

[1] Vegrad Flovik, "What is Graph Theory, and why should you care?" , August, 2020.

[2] Serdar Yegulalp , "What is TensorFlow? The machine learning library explained" , InfoWorld, 2019.

[3] Joseph Nelson, "What is ONNX?", RoboFlow, 2021.

[4] Yves Robert, "Task Graph Scheduling", France, 2011.

[5] Matthias Boehm, B. Reinwald, D. Hutchison, A. Evfimievski, P. Sen, " On Optimizing Operator Fusion Plans for Large-Scale Machine Learning in SystemML", 2018.

[6] Diederik Kingma, Jimmy Ba, "Adam: A Method for Stochastic Optimization" , 2014.

[7] Sachin Sharma, "A Comprehensive Case-Study of GraphSage using PyTorchGeometric and Open-Graph-Benchmark", 2021.

[8] Chinmay Jog, "The Two Benefits of the ONNX Library for ML Models",2020.

[9] Walker Rowe, Jonathan Johnson, "Machine Learning Frameworks To Use", 2020.

[10] Amal Menzli,"Graph Neural Network and Some of GNN Applications: Everything You Need to Know",2021.

[11] Janakiram MSV, "Open Neural Network Exchange Brings Interoperability to Machine Learning Frameworks" , 2020.

[12] Janakiram MSV,"Using a Pre-Trained ONNX Model for Inferencing",2020.

[13] Keith Pijanowski, "Making TensorFlow Models Portable Using ONNX" , 2020.

[14] Sherlock Huang, "ONNX Runtime Training Technical Deep Dive", 2020.

[15] Keyulu Xu, Mozhi Zhang, Stefanie Jegelka, Kenji Kawaguchi, "Optimization of Graph Neural Networks: Implicit Acceleration by Skip Connections and More Depth", 2021

PLAGIARISM REPORT