# BOOSTING PARTIAL CHANNEL NEURAL ARCHITECTURE SEARCH WITH GRADIENT PROJECTION

An Undergraduate Research Scholars Thesis

by

RYAN KING

Submitted to the LAUNCH: Undergraduate Research office at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by
Faculty Research Advisors: Dr. Bobak Mortazavi
Dr. Zhangyang Wang

May 2021

Major: Computer Science

# RESEARCH COMPLIANCE CERTIFICATION

Research activities involving the use of human subjects, vertebrate animals, and/or biohazards must be reviewed and approved by the appropriate Texas A&M University regulatory research committee (i.e., IRB, IACUC, IBC) before the activity can commence. This requirement applies to activities conducted at Texas A&M and to activities conducted at non-Texas A&M facilities or institutions. In both cases, students are responsible for working with the relevant Texas A&M research compliance program to ensure and document that all Texas A&M compliance obligations are met before the study begins.

I, Ryan King, certify that all research compliance requirements related to this Undergraduate Research Scholars thesis have been addressed with my Research Faculty Dr Bobak Mortazavi prior to the collection of any data used in this final thesis submission.

This project did not require approval from the Texas A&M University Research Compliance & Biosafety office.

# TABLE OF CONTENTS

# ABSTRACT

Boosting Partial Channel Neural Architecture Search with Gradient Projection

Ryan King
Department of Computer Science and Engineering
Texas A&M University

Research Faculty Advisor: Dr Bobak Mortazavi
Department of Computer Science and Engineering
Texas A&M University

Research Faculty Advisor: Dr Zhangyang Wang
Electrical and Computer Engineering
University of Texas at Austin

Neural Architecture Search has led to the discovery of novel neural network architectures that are capable of outperforming expertly designed architectures with fewer resource requirements at deployment time. This has lead to high performing neural networks that are small enough to fit into embedded systems and mobile devices. Recently, methods have been developed to significantly reduce the computational resources and time required to derive custom neural architectures. Specifically, gradient based methods have leveraged backpropagation to design architectures while a network is being trained, reducing search time from nearly 1400 GPU days to 1. However, differentiable neural architecture search suffers from dominating parameterless operations, steep local minimums, and shallow architectures. A recent multitasking method was able to reduce gradient confliction, dominating gradients, and high curvatures within their domain by projecting conflicting gradients from each task onto each other. We utilize a similar method to project conflicting gradients of edges in a search cell. In this paper we test various methods of gradient projection to determine the best way to avoid the derivation of suboptimal architectures. We show that differ-

entiable neural architecture search can be boosted with the use of gradient projection and partial channel connections. By doing so, we show that parameterless operations and steep local minimum can be related to dominating gradients and high curvatures that are overcome in the multitask setting.

# DEDICATION

*To my family, instructors and friends who have supported me throughout this process*

# ACKNOWLEDGMENTS


## Contributors

I would like to thank my faculty advisor, Dr. Bobak Mortazavi, and Dr Zhangyang Wang, for their guidance and support throughout the course of this research.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience.

Finally, thanks to my family for their encouragement and to my fiance for her patience and love.

The data used for Boosting Partial Channel Neural Architecture Search with Gradient Projection is a publicly available dataset provided by the Canadian Institute For Advanced Research. The analyses depicted in Boosting Partial Channel Neural Architecture Search with Gradient Projection were conducted in part by the Systems and Technology for Medicine and IoT (STMI) Lab and were published in 2021.

All other work conducted for the thesis was completed by the student independently.

## Funding Sources

No funding was received for this project.

# NOMENCLATURE

| | |
|---|---|
| MLP | Multilayered Perceptron |
| CNN | Convolutional Neural Network |
| RNN | Recurrent Neural Network |
| DAG | Directed Acyclical Graph |
| NAS | Neural Architecture Search |
| AutoML | Automated Machine Learning |
| SOTA | State of the Art |

# 1. INTRODUCTION

Neural Networks are machine learning models that have become popular due to their ability to perform complex tasks. A fully connect neural networks consists of multiple layers of parameterized perceptrons that perform a matrix transformation of their inputs. With a task such as classification or segmentation and an objective, the parameters of these layers can be trained to minimize the loss and increase the model's performance on its specific task. This enable users of neural networks to tune many parameters in a model to best fit a dataset with little domain knowledge.

Many advancements have been made in the past decade to adapt the standard neural network to various domains, such as Computer Vision and Natural Language Processing, and tasks, such as multitask and self-supervised learning. Some of the most impactful improvements have come from neural architecture design. The architecture of a neural network is represented by a set of hyperparameters that determine the design of the neural network. Some hyperparameters include the number of layers, the width of each layer, the type of operations used and the associated hyperparameters for each operation. However, high performing neural architectures have taken years for experts to design by hand for their specific domains.

Neural Architectures Search (NAS) is a field of Automated Machine Learning (AutoML) that attempts to automatically find high performing neural architectures. Due to it's ability to produce high performing lightweight neural architectures, NAS has become very popular for embedded systems and mobile devices. Until recently, NAS was computationally intensive due to the high number of hyperparameters that needed to be tuned along side the parameters in the architecture's operations. Most NAS methods used today utilize weight sharing and cell-based searches to reduce the number of hyperparameters that represent the architecture of the neural network. This has resulted in a reduction in search cost from 4000 GPU days to less than 1.

However, methods that have been used to reduce the computational cost of deriving an

architecture, have lead to many issues. In this paper, we review recent NAS methods that have suffered from dominating parameterless operations and shallow architectures. We provide a potential explanation for why these issue might be occurring. In addition, we use a multi-task learning method called gradient projection to help alleviate the issues we identify. We show that with this method, we are able to derive architectures that have comparable results to SOTA methods on the Cifar-10 and Cifar-100 datasets with half the computational cost.

# 2. RELATED WORK

## 2.1 Perceptron

The standard neural network builds from the concept of the perceptron. The perceptron model was developed by Frank Rosenblatt in 1958. Perceptrons are supervised models that perform binary classification. They accomplish this with a set of weights $W$ that are associated with each input feature from the data. With these weights, input data $X$ undergoes the following mapping:

$$f\left(\sum_{i}^{n} x_i w_i + b\right)$$

Where $x_i$ are the features of the input data, $w_i$ are the weights associated with each edge of the perceptron, $b$ is a bias parameter and $f$ is the activations function. In the original perceptron model, the activation function is the step function which is defined as:

$$f = \begin{cases} 0 \text{ if } 0 > x \\ 1 \text{ if } x \geq 0 \end{cases}$$

Here the activation function forces the perceptron to make a determination about the presence of an instance by producing a 1 for positive events and a 0 for negative events. A set of labels associated with the input data can be used to update the weight of the model and fit it to the data using the Parameter Learning Rule. The update is calculated as follows:

$$\Delta w_i = \eta(y_i - \hat{y}_i)x_i$$

Where $\eta$ is the learning rate, $y_i$ is the true label and $\hat{y}_i$ is the predicted label. The learning rate is a hyperparameter that effects the rate of change in weights. This equation is applied to each weight after each input sample until the weights have converged or changes become acceptable.

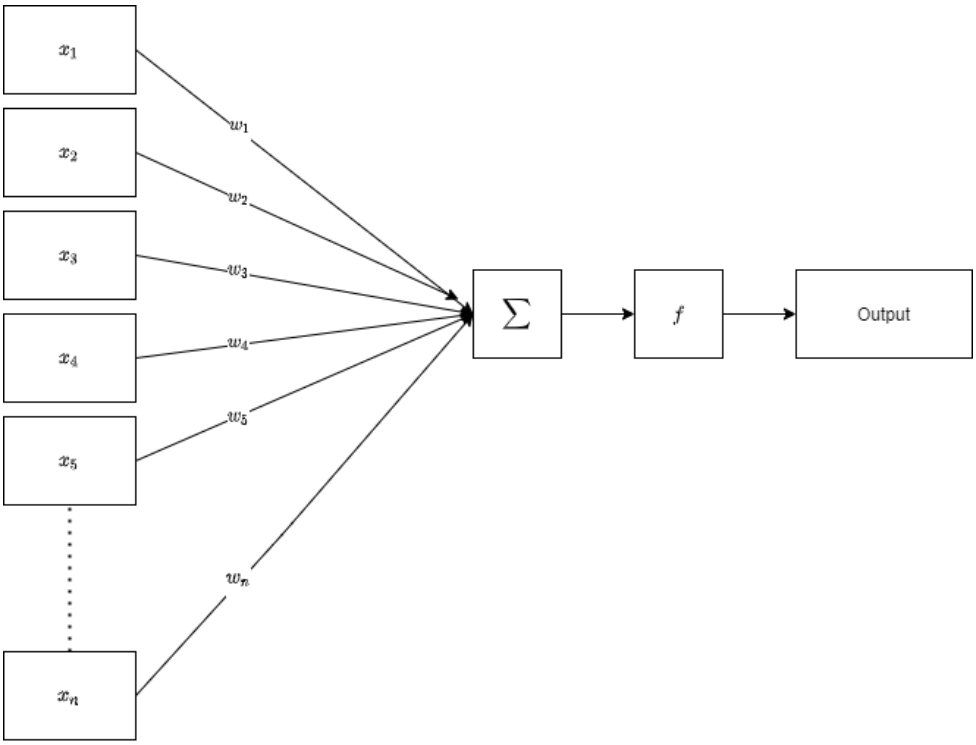


Figure 2.1: An illustration of the architecture of a perceptron.

## 2.2 Multilayered Perceptron

One major issue faced by the perceptron was that it could only classify data that was linearly separable. This is due to the lack of nonlinear transformations within the model. Two steps would be taken to transform the perceptron into a Neural Network. The first was the addition of nonlinear activation functions. A list of activation functions is provided in Appendix A. The second modification was the addition of multiple layers of different sizes, known as the Multilayered Perceptron (MLP). Each layers of the MLP consists of multiple perceptrons that transformed each input from the previous layer. Layers could be stacked provided the model with a hierarchical method of representing the data. The strength of this new model was proven by George Cybenko in 1989 when he showed that an MLP with two layers and a sigmoid activation function could approximate any function with his proof of the Universal Approximation Theorem. Recently, in 2017, it was proven that a MLP with bounded hidden layer sizes could be a universal approxima-

tor if it had the correct number of layers [1]. While both of these papers have proven that a well design neural architecture is powerful, there is no good method for determining the correct neural architecture to use.

### 2.3 Neural Architecture for Computer Vision

Since the first Neural Network, many operations and structures have been created that have drastically improved the performance of Neural Networks on many machine learning tasks. One of the first was in 1995, when LeNet-5 [2] became the first Neural Network to use a convolutional operation in their architecture. Much later in 2010, a GPU implementation of Neural Networks [3] was developed resulting in a major reduction in time spent training large Neural Network. In 2012, AlexNet [4] would build off the GPU implementation, along with deeper architecture, to achieve SOTA in classification of the ImageNet dataset with an increase of $12\%$ accuracy over the previous best. The major contribution from this work to neural architecture was the implementation of ReLU activations instead of the Hyperbolic Tangent and Sigmoid. This helped to reduce the effect of vanishing gradients which become more common as Neural Networks grew deeper. This allowed AlexNet to utilize 5 convolutional layers and 3 fully connected layers while LeNet-5 only used 2 fully connected layers with 3 fully connected layers.

The trend of creating Neural Networks that were deeper continued with VGG [5] which could be between 11 to 19 layers with the help of the 1x1 convolutional layer. ResNet [6] would build off VGG with skip connections. Skip connections would pass input forward by two layers, allowing their model to be up to 34 layers deep. While skip connections allowed neural networks to achieve SOTA, Andreas Veit et al. [7] conducted a lesion study on residual networks to determine if the longest paths were the largest contributors to the performance of the model. They found that paths through the network that contributed the most to the performance were relatively shallow. To overcome this issue, FractalNet [8] presented a method of dropping a path in a residual block during training while allowing other paths to train. This method proved to be a highly effective regularization strategy that reduced co-adaptability between operations within the residual blocks. Residual blocks are still used today along with many new structures and operations that reduce the

effects of vanishing gradients and allow for more powerful models.

## 2.4 Neural Architecture Search

Neural Architectures have proven to be a key component in the performance of Neural Networks. However, well designed neural architectures have taken decades to develop. Recently, a field of Automated Machine Learning (AutoML) called Neural Architecture Search (NAS) has emerged as a attempt to develop Neural Architectures automatically. One of the first NAS methods was NEAT [9] which used an evolutionary algorithm to develop Neural Architectures. NEAT represented Neural Networks as a Directed Acyclical Graph (DAG) and the operations as a set of edges along the graph. A set of Neural Networks would then be created and trained on a dataset. The graphs of the best performing Neural Networks would then be used to created offspring Neural Networks. While this method at the time provide 5 times increase in speed, the method involved evaluating thousands of networks.

### *2.4.1 Cell Based Neural Architecture Search*

NASNet [10] proposed a method of simplifying the search method with cell based NAS. Instead of searching for entire neural architectures, cell based NAS reduced the problem to search for the architectures of a few repeatable Residual Blocks called cells. In NASNet, two types of cells were used: the normal cell output the transformed data with the same dimension as their input; reduction cell output the transformed data at half the size and twice as many channels. With this formulation of NAS, only the architecture of the two cells needed to be embedded. The cell is represented as a DAG where the vertices represent each step in the cell and the edges represent the operations being performed between steps. At each step, the output of the operations from each edge are summed together. Finally, each step in the cell is concatenate together to form the output of the cell. An example of a cell architecture is given in Figure 2.2. Each cell is stacked in a chain to create the super structure of the model.

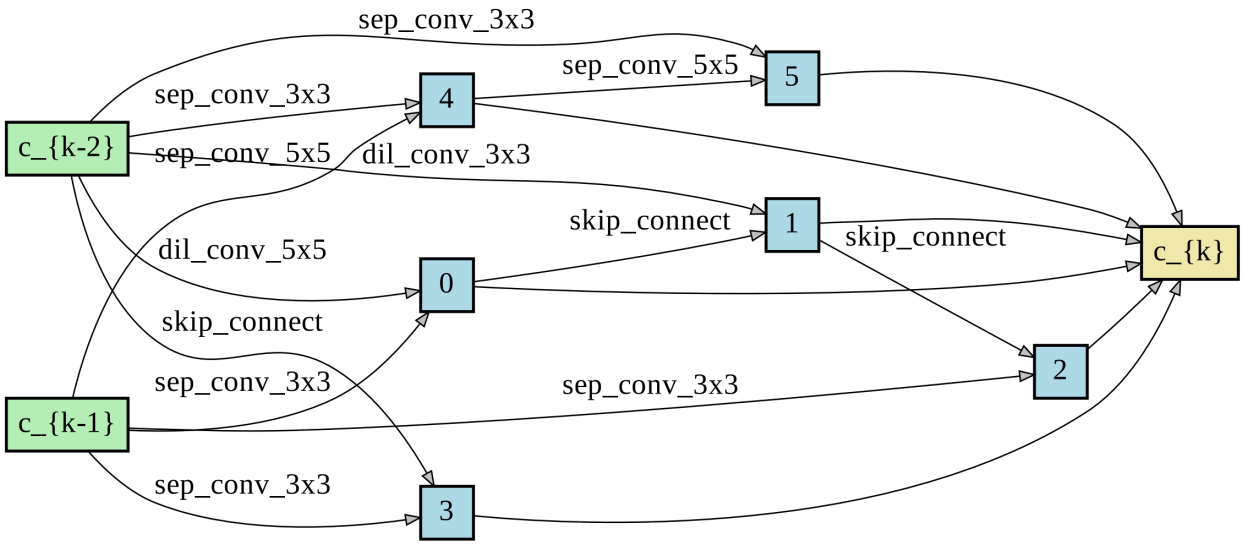


Figure 2.2: An example of the architecture of a cell with 6 steps. Each step in this cell takes two inputs from previous stesp in the cell.

In NASNet, the architecture of the cells is created by a controller Recurrent Neural Network (RNN). The controller network uses the performance of an evaluated architecture to created the edges between steps and the operations along those edges. The controller network is able to choose operations from a set of candidate operations that are predefined. A list of the candidate operations found in NASNet and other methods discussed in this paper can be found in Appendix A. This method was able to create models that achieved SOTA performance at the time with nearly half the parameters of handcrafted models. NASNet's biggest shortcoming was that it took 2000 GPU days in order to create that architecture.

### *2.4.2 Differentiable Neural Architecture Search*

DARTS [11] provided a solution to this issue with a continuous relaxation of the operation search space. Using the same structure from NASNet, DARTS replaced the edges of the cell architecture with a mixture of operations, $\hat{\mathcal{O}}$. For each mixture of operations, each operation, $o$, in the DARTS search space received the inputs of the previous steps. The output of those operations

Table 2.1: Accuracy of various NAS methods derived and evaluated using CIFAR-10

| | Top-1 Error | Search Cost (GPU Days) | Model Size |
|---|---|---|---|
| DARTS V1 [11] | 3.00 | 1.5 | 3.3 |
| DARTS V2 [11] | 2.76 | 4 | 3.3 |
| GDAS [13] | 2.82 | 0.17 | 2.5 |
| PDARTS [14] | 2.50 | 0.3 | 3.4 |
| PC-DARTS [15] | 2.57 | 0.1 | 3.6 |

were then multiplied by a weight that represented that operations importance along that edge. A set of architecture parameters, $\alpha$, are used to determine the weights for each operation at each edge by taking the softmax of the weights along each edge. Each mixture of operations is represented by:

$$\hat{\mathcal{O}}_{i,j}(x_i) = \sum_{o \in \mathcal{O}} \frac{exp\left(\alpha^{o}_{(i,j)}\right)}{\sum_{o' \in \mathcal{O}} exp\left(\alpha^{o'}_{(i,j)}\right)} o(x_i)$$

By incorporating the embedded architecture into the forward pass of the Neural Network, backpropagation could be used to derive the correct weight for the architecture. A bi-level optimization [12] is used to train the architecture parameters and network parameters jointly. By making the architecture parameter differentiable, DARTS was able to achieve the same performance as NASNet in 1.5 GPU days.

### *2.4.3 Steep Local Minimum*

Gradient based NAS has proven to be an efficient method of deriving good Neural Architectures. However, Arber Zela et al. [16] identified an overfitting issue that was being faced by the method. The architecture parameters of gradient based NAS were converging to sharp local minimums. This became an issue when model edges were pruned between the architecture derivation and evaluation of the architecture. Due to the sharp convergence of the architecture parameters, the slightest change in the architecture resulted in drastic performance decreases. RobustDARTS [16] provide a form of early stopping based on the eigenvalue of the Hessian martix of the architecture

parameters. This enabled RobustDARTS to stop architecture training before reaching an area that was too sharp. SmoothDARTS [17] additionally, attempted to address this issue by forcing the architecture optimizer to consider the neighborhood of the current architecture parameters.

### *2.4.4 Dominate Parameter-less Operations*

RobustDARTS [16] additionally pointed out that parameter-less operations such as skip connections, max pooling and average pooling were dominating edges in cases where they obviously harmed performance. This was due to the unhindered gradient flow through parameter-less operations. Many methods have proposed solutions to this problem including PDARTS [14], PC-DARTS [15] and NoisyDARTS [18].

PDARTS [14] attempts to reduce this affect by progressively reducing the number of operations in the search space, while increasing the number of layers to provide a search model that more closely represents the final architecture. The final step of PDARTS, directly addresses this parameter-less operation issue by restricting the number of skip connections in the final architecture.

PC-DARTS [15] uses a combination of partial channel connections and an extra set of architecture parameters that are used for edge normalization. The partial channel connections used in PC-DARTS use a random fraction of the channels in convolution operations from the NAS search space. By doing this, they speed up the derivation of architectures while regularizing the model. PC-DARTS additionally attempts to reduce the uncertainty of their architecture searches with a set of architecture parameters that determine the weight of each edge in the search cell.

### *2.4.5 Gradient Projection*

In multitask learning, a neural network is trained on multiple tasks at the same time. This is done with a set of subnetworks for each task that branch off of the main network. A recent method in multitask learning [19], called Gradient Surgery, identified three issues within the multitask learning domain that degrade the performance models. As the gradients from each head of the multitask network converge on the main neural network, gradients begin to suffer from issues that

they call the tragic triad. Those issues are conflicting gradients, dominating gradients and high curvatures. Similar to the measure used in RobustDARTS [16], high curvature is measured the same way and negatively effects training. Dominating gradients occur when one of the gradients from a task is significantly larger than the gradients of other tasks. Gradients that are dominated by larger gradients can have negligible contributions to training in these cases. Gradient confliction occurs when the incoming gradients from each lack similarity. Similarity in this case is the cosine-similarity of the gradients which is:

$$cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|}$$

where $\mathbf{A}$ and $\mathbf{B}$ are the gradients of incoming tasks. If the gradients point in conflicting directions, the cosine similarity is less than 0. Gradient Surgery addresses these issues by measuring the similarity between gradients during each backwards pass. If gradients are conflicting, a random gradient is projected onto the normal vector of the respective conficting gradient. In doing so, Gradient Surgery eliminates gradient confliction and alleviates the effects of high curvature and dominating gradient.

### *2.4.6 Proposal*

We hypothesize that the gradients flowing through parameterless operations and shallow edge in a differentiable NAS search cell, are dominating all other potential edges and operations. We believe that by correcting the gradients, the performance of the derived architecture can be improved. We attempt to alleviate these issues with gradient projection due to it's ability to correct gradients with similar issues. We test the gradients of the architecture parameters flowing in and out of each step in a search cell for confliction. Conflicting gradients are then projected onto the normal of it's associated conflicting gradient. Similar to the use of gradient projection in the multi-task setting, we hope that by projecting conflicting gradients, we can reduce the effects of dominating gradients in differentiable NAS.

# 3. METHODS

We use the PyTorch [20] library to develop our model and this method. Since we are testing the effects of projecting conflicting gradients in differentiable NAS with partial channel connections, we only change the architecture step when deriving neural architectures. An illustration of how our architecture model is trained can be seen in Figure 3.1.

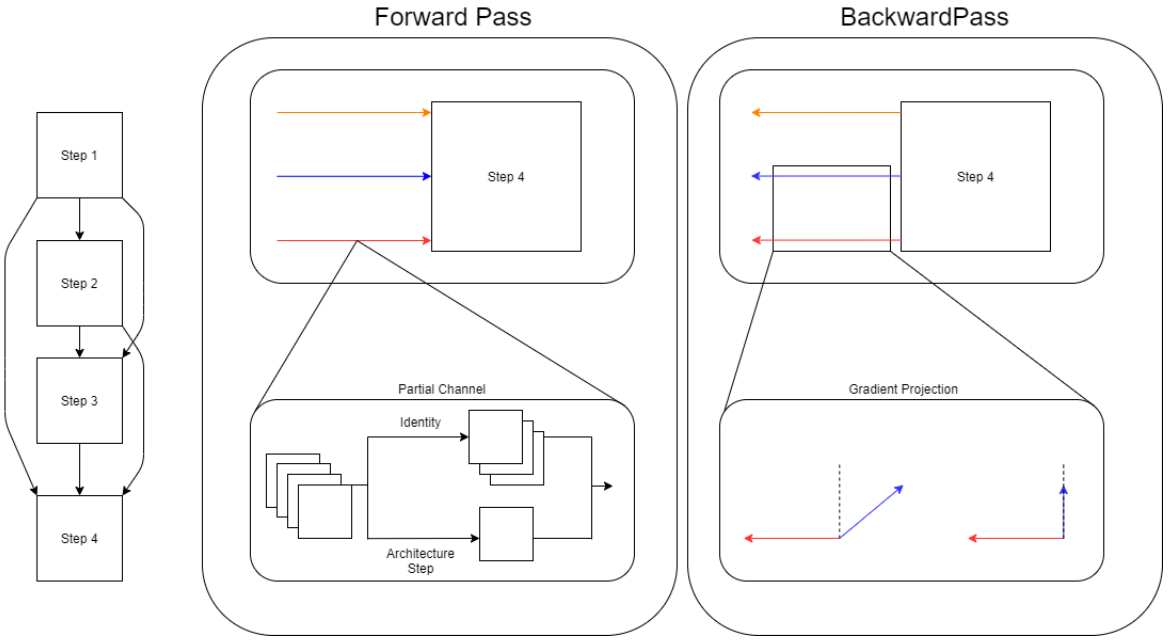


Figure 3.1: The pipeline that we outline in this paper. On the far left is the structure of a search cell. The center image depicts the partial channel connections during the forward pass through the network. The image on the right depicts the projection of conflicting gradients during the backwards pass through the network.

## 3.1 Architecture Training

We follow previous gradient based neural architecture search methods and use a bi-level optimization to train our model and architecture parameters simultaneously. The first level of the optimization updates the parameters of the operations in the model. The second level of the

optimization updates the embedded architecture parameters. In order to avoid overfitting between the model and the architecture parameters, the datasets used for training are split to allow half of the data to be used for standard model training while the other half is used for the architecture updates. Following current differentiable NAS methods, we use cutout [21], auxiliary towers [22] and drop path [8] to boost the training of our model. During architecture training and evaluation, we use a cosine annealing learning rate scheduler [23]. A full list of the parameters used in architecture training and evaluation can be found in Appendix B.

### 3.2 Datasets

We use 2 different image datasets to train and evaluate our models: CIFAR-10 [24] and CIFAR-100 [25]. CIFAR-10 is an image dataset that consists of 60,000 32 by 32 color images with 10 classes. We use 50,000 of those images for training and 10,000 for testing. CIFAR-100 contains the same number, shape and size of images as CIFAR-10 but it contains 100 classes. We use the same split for this dataset as we use for CIFAR-10. We use these datasets to compare with previous related results and to test the effectiveness of our method on datasets with varying degrees of difficulties.

### 3.3 Gradient Projection

In order to identify issues that can cause parameter-less operations to become dominate, we measure the similarity between the edges of a step. We train six different architectures using different projection methods to determine which produces the best performance. Projection methods include the following:

- Project the shortest path onto the longest
- Project the longest path onto the shortest
- Project edges at random

The shortness and longness of an edge is determine by the maximum number of edges that the data can pass through if an edge is used. A labeled illustration is shown below:

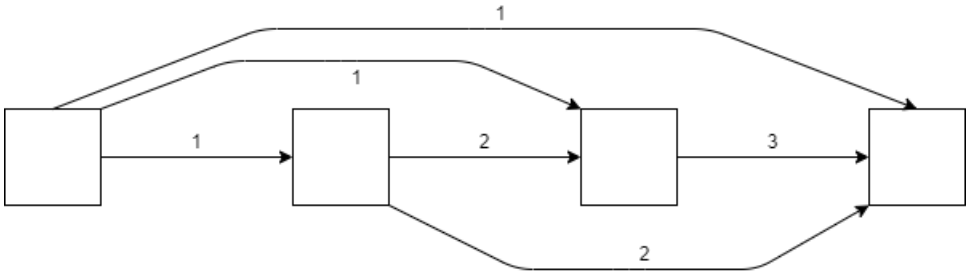


Figure 3.2: An illustration of the edge lengths within a cell. The number labels for each edge represent the longest path that the data can take to get to the next step from that edge.

We test these three projection methods for incoming and outgoing edges making a total of six different projection methods. The motivation for projecting edges based on edge path length comes from Andreas Veit et al. [7] which shows that the majority of gradient flow goes through the shallowest path in a network. We hope that by projecting paths based on their length, we can directly address the issue of shallow architectures in differentiable NAS. We use the Cifar-100 dataset to derive the architectures for these methods and evaluate the resulting architecture. Results of these tests are found in Table 4.1.

### 3.4 Partial Channel Connections

We use partial channel connections for all convolutional operations in our search space. We use $\frac{1}{4}$ of the channels in each convolutional layer while the rest of the channels are passed forward with no transformation. At the end of each forward pass, channels are shuffled to ensure that during the next pass, a new random sample is selected.

### 3.5 Edge Normalization

We use an additional set of architecture parameters for edge normalizing. Each parameter is associated with a specific edge in each type of search cell. This weight allows for decoupled training between the operations along the edges and the strength of each edge. During the architecture step of our training, both sets of architecture parameters are updated.

# 4. RESULTS

## 4.1 Projection Methods

We train six different architectures using the different projection methods we discuss in Section 3.3 to determine which method produces the best performance. We record the top-1 error of each architecture evaluated on the CIFAR-100 dataset. Table 4.1 contains the results from those tests. Table 4.2 contains a comparison of various differentiable NAS methods and our method evaluated on CIFAR-100.

Table 4.1: Accuracy of various methods of projecting conflicting gradients.

| | Top-1 Error | Model Size |
|---|---|---|
| Edge-In Shortest to Longest | 17.2 | 4.16 |
| Edge-In Longest to Shortest | 17.5 | 4.08 |
| Edge-In Random Projection | 16.2 | 4.42 |
| Edge-Out Shortest to Longest | 17.41 | 4.27 |
| Edge-Out Longest to Shortest | 17.62 | 3.95 |
| Edge-Out Random Projection | 16.69 | 4.25 |

The results from each projection method show that random projection performs best in both the incoming and outgoing edge scenario. This is consistent with Gradient Surgery [19] where incoming edges from multiple tasks are randomly projected. This indicates that projection of conflicting gradients does not favor methods that project onto shortest or longest paths. By randomly projecting, the gradient in both the shortest and longest paths are allowed to learn with an equal amount of correction from gradient projection.

In Figure 4.1 we show the architecture of both the normal and reduction cell from the best projection method. Figure 4.1 contains the resulting architecture of the best normal and reduction cell from the best projection method. Cell architectures derived with this projection method are deeper than PC-DARTS and DARTS with fewer connections to shallower steps. The architecture

also avoids the use of parameterless operations without the use of a direct restriction as in PDARTS [14].

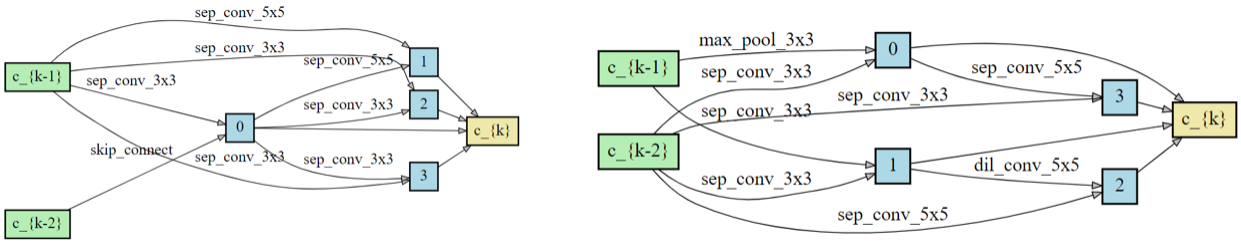


Figure 4.1: The normal (left) and reduction (right) cell architectures for the top performing projection method on Cifar 100.

In table 4.2, we compare the results of our method on Cifar-100 with comparable differentiable NAS methods. Our method achieves comparable results with the SOTA differentiable NAS method in half the time.

Table 4.2: A comparison of the results from Gradient Projection and other related NAS methods trained and evaluated on CIFAR-100. * Architecture derived on CIFAR-10.

| | Top-1 Error | Search Cost (GPU days) | Model Size |
|---|---|---|---|
| GDAS [13] + cutout | 18.13* | 0.17 | 2.5 |
| PDARTS [14] + cutout | 15.92 | 0.3 | 3.6 |
| PC-DARTS [15] + cutout | 17.26 | 0.1 | 3.1 |
| PC-DARTS + Grad Projection + cutout | 16.2 | 0.15 | 4.42 |

## 4.2 Results of CIFAR-10

Table 4.3 contains a comparison of various differentiable NAS methods and our method evaluated on CIFAR-10. The results from our test on Cifar 10 show that our method achieves comparable results with the SOTA differentiable NAS method with the resulting architecture being derived in half the time.

Table 4.3: A comparison of the Top-1 Error (lower is better) from Gradient Projection and other related NAS methods trained and evaluated on CIFAR-10. Results from our method are the average of 5 runs.

| | Top-1 Error | Search Cost (GPU days) | Model Size |
|---|---|---|---|
| DARTS V1 [11] + cutout | 3.00 | 1.5 | 3.3 |
| DARTS V2 [11] + cutout | 2.76 | 4 | 3.3 |
| GDAS [13] + cutout | 2.82 | 0.17 | 2.5 |
| PDARTS [14] + cutout | 2.50 | 0.3 | 3.4 |
| PC-DARTS [15] + cutout | 2.57 | 0.1 | 3.6 |
| PC-DARTS + Grad Projection + cutout | 2.48 | 0.14 | 3.9 |

Figure 4.2 shows the resulting architecture of the best architecture derived on Cifar 10. Similar to the architecture derived on Cifar 100, the edges of this architecture have deeper connections with few parameterless operations. Most steps in the architecture favor having one shallow connection along with a deep connection. This scheme is seen in both the Cifar 10 and Cifar 100 architectures. This construction of steps within the cell acts similar to a skip connection by allowing gradients to flow through the network while developing steps that are deeper.

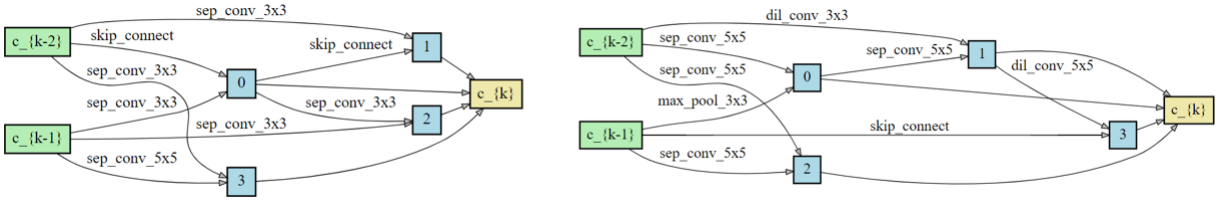


Figure 4.2: The normal (left) and reduction (right) cell architectures for the top performing projection method on Cifar 10.

# 5. CONCLUSION

## 5.1 Differentiable Neural Architecture Search

In this paper, we discuss some of the issues that hinder differentiable NAS performance. Specifically, we focus on two problems that have been proven to exist is both differentiable NAS and in neural architectures. The first problem we attempt to address is the problem of parameterless operations such as skip connections, average pooling, and max pooling. Since these operations are parameterless, gradient flows freely through those operations resulting in suboptimal architectures. The second problem we address is the tendency for gradients to flow through the shallowest path in a multi-branch network. We hypothesize that gradients flowing through parameterless operations and shallow edges dominate other operations and edges, giving them no chance to develop.

## 5.2 Gradient Projection

We attempt to directly address this issue by using a method called gradient projection on the architecture parameters of the search model. This method has been used in multi-task learning to boost performance and reduce the effects of dominating gradients. We test for conflicting gradients between edges associated with steps in the search cell. If confliction is detected, one of the conflicting edges is projected onto the normal vector of the other. We test various methods including the projection of the shallowest edge onto the deepest, deepest to shallowest and random projections. In addition we test projection on incoming edges along with the outgoing edges.

## 5.3 Results

We find that by randomly projecting the architecture gradients of edges going into a step within a search cell, we can boost performance with a slight increase in time taken to develop architectures. Even with a slight increase in the time taken to derive architectures, the performance gain from our method achieve comparable results with the SOTA differentiable NAS method in half the search time.

# REFERENCES


[1] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang, "The expressive power of neural networks: A view from the width," *CoRR*, vol. abs/1709.02540, 2017.

[2] Y. LeCun *et al.*, "Lenet-5, convolutional neural networks," *URL: http://yann. lecun. com/exdb/lenet*, vol. 20, no. 5, p. 14, 2015.

[3] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep big simple neural nets excel on handwritten digit recognition," *CoRR*, vol. abs/1003.0358, 2010.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), vol. 25, pp. 1097–1105, Curran Associates, Inc., 2012.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[7] A. Veit, M. J. Wilber, and S. J. Belongie, "Residual networks are exponential ensembles of relatively shallow networks," *CoRR*, vol. abs/1605.06431, 2016.

[8] G. Larsson, M. Maire, and G. Shakhnarovich, "Fractalnet: Ultra-deep neural networks without residuals," *CoRR*, vol. abs/1605.07648, 2016.

[9] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary computation*, vol. 10, no. 2, pp. 99–127, 2002.

[10] B. Zoph, V. Vasudevan, J. Shlens, and Q. V. Le, "Learning transferable architectures for scalable image recognition," *CoRR*, vol. abs/1707.07012, 2017.

[11] H. Liu, K. Simonyan, and Y. Yang, "DARTS: differentiable architecture search," *CoRR*, vol. abs/1806.09055, 2018.

[12] B. Colson, P. Marcotte, and G. Savard, "An overview of bilevel optimization," *Annals of operations research*, vol. 153, no. 1, pp. 235–256, 2007.

[13] X. Dong and Y. Yang, "Searching for A robust neural architecture in four GPU hours," *CoRR*, vol. abs/1910.04465, 2019.

[14] X. Chen, L. Xie, J. Wu, and Q. Tian, "Progressive differentiable architecture search: Bridging the depth gap between search and evaluation," *CoRR*, vol. abs/1904.12760, 2019.

[15] Y. Xu, L. Xie, X. Zhang, X. Chen, G. Qi, Q. Tian, and H. Xiong, "PC-DARTS: partial channel connections for memory-efficient differentiable architecture search," *CoRR*, vol. abs/1907.05737, 2019.

[16] A. Zela, T. Elsken, T. Saikia, Y. Marrakchi, T. Brox, and F. Hutter, "Understanding and robustifying differentiable architecture search," *CoRR*, vol. abs/1909.09656, 2019.

[17] X. Chen and C.-J. Hsieh, "Stabilizing differentiable architecture search via perturbation-based regularization," 2020.

[18] X. Chu, B. Zhang, and X. Li, "Noisy differentiable architecture search," 2020.

[19] T. Yu, S. Kumar, A. Gupta, S. Levine, K. Hausman, and C. Finn, "Gradient surgery for multi-task learning," 2020.

[20] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[21] T. Devries and G. W. Taylor, "Improved regularization of convolutional neural networks with cutout," *CoRR*, vol. abs/1708.04552, 2017.

[22] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *CoRR*, vol. abs/1409.4842, 2014.

[23] I. Loshchilov and F. Hutter, "SGDR: stochastic gradient descent with restarts," *CoRR*, vol. abs/1608.03983, 2016.

[24] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research),"

[25] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-100 (canadian institute for advanced research),"

# APPENDIX A: Activation Functions

| | NASNet | DARTS | NAS Bench 201 |
|---|---|---|---|
| Zero | | x | x |
| Identity | x | x | x |
| 1x7 then 7x1 Conv | x | | |
| 1x3 then 3x1 Conv | x | | |
| 3x3 Average Pooling | x | x | x |
| 3x3 Max Pooling | x | x | |
| 5x5 Max Pooling | x | | |
| 7x7 Max Pooling | x | | |
| 1x1 Convolution | x | | x |
| 3x3 Convolution | x | | x |
| 3x3 Depthwise-Separable Convolution | x | x | |
| 5x5 Depthwise-Separable Convolution | x | x | |
| 7x7 Depthwise-Separable Convolution | x | | |
| 3x3 Dilated Separable Convolution | | x | |
| 5x5 Dilated Separable Convolution | | x | |

## APPENDIX B: Activation Functions

Architecture Derivation Configuration

- Architecture Learning Rate = 0.0006
- Architecture Weight Decay = 0.001
- Batch Size = 256
- Epochs = 50
- Initial Channel = 16
- Layers = 8
- Learning Rate = 0.1
- Learning Rate Minimum = 0.0
- Momentum = 0.9
- Weight Decay = 0.0003

Evaluation Configuration

- Auxiliary Weight = 0.4
- Batch Size = 96
- Cutout Length = 16
- Initial Channel = 36
- Layers = 20
- Learning Rate = 0.025

- Momentum = 0.9
- Weight Decay = 0.0003