Empirical Study on convergence of Capsule Networks with various hyperparameters

Abhishek Chauhan, Mukund Babu, Nischel Kandru, Sanket Lokegaonkar
Project Group 7

***Abstract*— With tremendous research growth in machine learning domains and specifically deep learning models, there has been a lot of advancements pertaining to this field in the recent times. Convolutional Neural Networks (CNNs) have had a huge impact on the Deep Learning field in recent times because of the complex problems it solves and its wide-range applications. We know that CNNs perform incredibly well on classifying images like the ones in the dataset; however, they fail to explore the spatial relationship between features and fail to classify different variants of the same image. Hinton et al proposed the Capsule Network with Dynamic Routing; which specifically addresses these issues with CNNs. This network utilizes Capsules; which detects the likeliness of a feature in an image and the orientation of the feature in that image. Hinton et al also proposed an enhanced Capsule network; Matrix Capsules with EM Routing. A Matrix Capsule contains a logistic unit to describe the likeliness of a feature and a 4*4 pose matrix to describe the relationship between that feature and the pose.**

In this research, we attempt to analyze the differences between CNN, Capsule Network with Dynamic Routing and Matrix Capsules with EM Routing from an optimization, convergence perspective where we analyze the performance of all the three models on different optimizers and configuration hyperparameters. We perform this experiment on different datasets like smallNORB, CIFAR10, and Tiny ImageNet.

INTRODUCTION

Machine Learning techniques have been very popular in recent times with the growing number of datasets and computation capabilities. Deep Learning Models have become very popular and are used in most of the image processing, computer vision problems, disease classification and NLP based problems like text classification, recommender systems. With its high impact on such critical applications; it is easily understandable that Neural Networks attracts many researchers. A lot of research is being carried out to increase the performance of these models and compute the best optimizers and hyperparameters for different datasets so that these models (when customized) deliver best results for all kinds of data.

Convolutional Neural Networks (CNNs) have been around for a long time with many innovations being carried out related to them. AlexNet, a recent improvement to Convolutional Neural Network has performed much better on the ImageNet dataset than any other previous CNN method. Even with such performances, CNNs have a couple of major issues surrounding them because CNNs just look for the existence of a feature in an image to classify it and not for the relative spatial location of the feature. This implies that CNN considers the cases where a feature isnt correctly aligned with respect to the other features in the image and a feature being correctly aligned to other features as the same classification. CNNs fail to explore the spatial relationship between features and fail to classify different variants of the same image. This deficiency is mainly because CNNs pass information across layers utilizing max pooling [7].

Capsule Networks introduced by Hilton et al in their paper titled Dynamic Routing Between Capsules has attempted to solve the deficiencies of CNNs. A capsule consists of a group of neurons whose output is a vector to signify the existence of an entity. This vector contains all the characteristics of the entity and its length signifies the probability that the entity exists. Each lower level capsule sends its output to all the higher-level capsules by multiplying the vector with a weight matrix which signifies its relationship with the higher level. The parent capsule that obtains the larger scalar product is activated. This Dynamic Routing wherein routing-by-mechanism is used; achieves state of the art performance [4], [5], [9], [10].

Hilton et al also described an enhanced version of Capsule Networks [8] where each capsule is described using a logistic unit to represent the presence of the entity and a matrix to describe the properties of the entity (pose matrix), which is a 4x4 matrix to represent different characteristics like spatial coordinates, orientation of the object and other characteristics of a feature. The connection between the lower capsule and parent capsule is a transformation matrix (also 4x4). They use the Expectation Maximization (EM) routing to group appropriate capsules in a lower layer to the higher layer to form a part-whole relationship [6], [8].

In this research, we perform an empirical study on the convergence of Matrix Capsules with EM Routing with various hyperparameters. We study the convergence of this model for different Optimizers like Adam, Adadelta, Adagrad and Rmsprop and vary different hyperparameters like number of channels in the first convolution layer (A), number of primary capsule layers (B), number of capsules in convolution layers following primary layer (C), number of capsules in the last convolution layer (D) and number of routing iterations (r). We perform this computation on different datasets like Cifar-10, smallNORB, TinyImageNet, and MNIST. We try to compute the best set of configurations which performs efficiently [11], [12].

We compare those results with the initial variant of Capsule Network which utilizes Dynamic Routing and the baseline CNN model for smallNorb with Adam, Adadelta, Adagrad, and Rmsprop optimizers to study the effect of

different optimizers. The Capsule Networks is the latest innovation in Neural Networks and there would be a lot of future research; so, this can be a great starting point for the research community. Researchers would have a better understanding of the optimizers that are suitable for different datasets and understand the impact of different hyperparameters. Our results deliver the best combination of hyperparameters which will lead to better convergence and accuracy. This is a vital study for future research on this model with different optimizers.

Our comparison with the baseline models like CNN and Capsule Networks with Dynamic Routing can help people understand the impact of different optimizers which leads to building up efficient optimizer parameters for these models.

BACKGROUND

In this section, we provide an overview of how a Convolutional Neural Networks work [3]. We discuss the shortcomings of CNNs which provide motivation to develop Capsule Networks. We will then discuss the functioning of a capsule network.

A. Convolutional Neural Network

Convolutional Neural Networks are a category of neural networks which has been proven very effective in image recognition and classification. CNNs are very popular because use relatively very little pre-processing as compared to other image classification algorithms. CNNs learn the filter that in other image classification algorithms are engineered manually. Each image is represented as a matrix of pixel values. A colour image will have three channels red, green, blue and the image will be represented in terms of 2d matrices for each of the channels containing values ranging from 0 to 255 according to each pixel. A grayscale image will have only channel which will a single 2d matrix and value of each pixel will range from 0 to 255. In this section, we will provide a basic overview of the functioning of a CNN. CNNs have four major operations:

1) Convolution Step: The primary function of the convolution step is the extraction of features from the input image. CNNs use a matrix called the filter which is slid across the input image matrix to compute the dot product between the two matrices to form the feature map. This feature map is what the CNN learns from the image. Different values of the filter matrix produce different feature maps and it is essential to define different filters to capture various features of the input image. A CNN learns the filter values themselves during the training process but we can specify the number of filters, filter size etc. before the training process.

2) Non-Linearity: After the convolution step, a Rectified Linear Unit step is performed which an element-wise operation to replace all the negative pixel values with zero. This is to introduce non-linearity since the real-world data will mostly be non-linear. There are other non-linear functions which can be used in this step but Rectified Linear Unit is found to perform well in most situations.

3) Pooling: Pooling is a process which reduces the dimensionality of the feature maps while retaining the relevant information. It is done on multiple ways: max, average, sum etc. Spatial pooling helps in making the feature representation smaller and manageable. It also reduces the parameters and computations to avoid the overfitting problem. It makes the CNN invariant to small transformation, distortions and translations.

4) Fully Connected Network: The fully connected layer is a traditional Multi-Layer Perceptron with the activation function in its output layer being a softmax function. Every neuron in the previous layer is connected to every neuron in the next layer. The output after the pooling step is a high level information about the features of the input image. The fully connected layer uses this information to classify the input images into corresponding classes based on the training dataset.

Training: Once the initial parameters are defined, the CNN takes the input image from the training dataset and goes through the forward propagation step (all the steps listed above). At the end, it computes the output probabilities for each of the input classes and the total error at the output step. Then it uses backpropagation to calculate the gradients of the error and use gradient descent to minimize the error by updating the filter matrix and initial parameters of the model. This is repeated for all the images in the training dataset to obtain the optimal value of all the weights and the parameters. Once this is obtained, the CNN will be able to classify any unseen image.

B. Capsule Networks

Convolutional Neural Networks works very well for image classification. However, CNNs perform exceptionally great when the classifying images are similar to the images in the training dataset. If there are translations, rotations to the images from the training dataset, the CNNs have a very poor performance. As mentioned in the section earlier, pooling helps in creating positional invariance to avoid overfitting. Pooling leads to false classifications of images which only have components of the original image but not the actual image itself. Equivariance is a process through which a CNN understands the rotation or translation of an image and adapt itself to classify accordingly. The effort to achieve equivariance led to the development of capsule networks. In this work, we have performed an empirical study based on the work of Hinton et. al [8] on capsule networks.

A capsule is a group of neurons whose outputs represent different properties of the same entity in the image. Each layer in the capsule network consists of many capsules. So capsule networks can be viewed nested neural networks. On a high level, the capsule networks,as described by Hinton et. al [8], works as follows: each capsule in a layer consists of a 4×4 pose matrix and an activation probability. In between each capsule in one layer to another capsule in the next layer is another 4×4 matrix which is the trainable transformation matrix. These matrices are learned by the network automatically and these are the only stored parameters.

The pose matrix of a capsule in one layer is transformed by this transformation matrix to cast a vote on the pose matrix of a capsule in the adjacent layer. This voting process for each of the capsules in a layer is determined by a non-linear dynamic routing procedure. In fact, the effectiveness of the capsule networks is greatly attributed to the dynamic routing process which captures the part-whole relationship between different entities in the image. The authors use a version of the Expectation-Maximization algorithm which iteratively updates the activation probabilities of capsules in the next layers and the assignment probabilities of the capsules in the previous layers.

Dynamic Routing: As mentioned before, dynamic routing is the process of routing outputs from layer L to a layer $L+1$ and this process replaces the concept of max pooling used in CNNs. The capsule network learns the transformation matrix corresponding to a connection a capsule from layer L to all the capsules in layer $L+1$. In addition to these transformation matrices, each connection is also multiplied by a dynamically computed routing co-efficient. A capsule in one layer of the network represents one entity in the image and this entity is part of a larger entity of the image in the capsule in the next layer. Part of the dynamic routing process is to figure out the probability with which a capsule in one layer is activated because of the entity in the next layer. Since the pose matrix captures all the spatial information related to all the features, a layer is able to agree with significant confidence that the activated capsules belong to one class of images. This is called routing by agreement. The routing co-efficient, which is used along with the transformation matrices to route outputs between layers, is not fixed and they learned with every forward pass of the algorithm.

EM for Routing by Agreement: Hilton et. al [8] look at the routing process between capsules of two layers as fitting a mixture of Gaussian using the EM algorithm. The higher level capsules act like the Gaussians and the pose matrix of each active capsules in the lower layer act as the datapoints. Given a capsule at level L, we have a choice to activate or not activate a capsule at the next level $L+1$. If we do not activate it, we incur a cost for each capsule at the lower level. And if we do activate a capsule at the higher level, we have a cost for coding its mean and variance. This works exactly like the EM algorithm, where the E-step is used to determine the probability of a datapoint to be assigned to a Gaussian and the M-step calculates the means of each Gaussian and the variance around the mean. The pose matrices and the activations of the capsules in layer $L+1$ is computed by running this algorithm for a few iterations. The authors called this process EM Routing.

Architecture & Implementation

Our focus with this work was to analyze the behavior of the modern adaptive optimization algorithms on recently proposed CapsNet architectures and contrast them with CNN architectures. For comparison, we compare to 3 different architectures:

a) CNN Architecture:: For baseline CNN architecture, we use 2 layer convolution model of 32, 64 channels. First channel has 5×5 kernel and stride of 1. The last convolution layer is followed by one fully connected layer which maps the kernel to the output class probabilities.

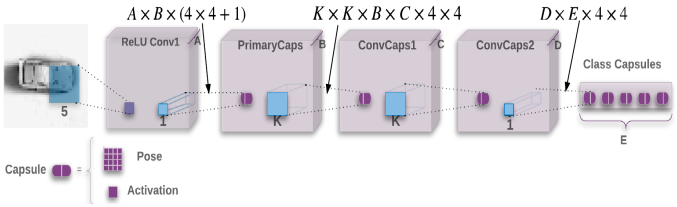

Fig. 1. Architecture of the Network

b) CapsuleNet-(Dynamic Routing):: For Dynamic Routing-based CapsNet architecture , we use a the architecture followed by the original paper. It consists of 2 convolution layers and one fully connected layer. The first convolution acts like a initial feature extractor and are passed as inputs to the primary capsules. The first convolution layer has 9×9 convolution kernels with a stride of 1 followed by ReLU activation. The primary capsules is a 32 channels of 8D convolution capsules and generate $[32 \times 6 \times 6]$ capsule output vectors of 8 dimensions. Dynamic Routing mechanism is introduced between the primary and secondary capsules. Each capsule vector of the lower layer is connected to the capsules on the higher layer. The final DigitCaps layer is mapped to the 16D capsule per class and trained on reconstruction loss and l2 logits loss.

c) Matrix Capsules:: We used the same architecture as described by Hinton et. al [8] as depicted the figure .1 The model starts with a 5×5 convolutional layer with **A** number of layers and 2 strides with a ReLU non-linearity. The other layers are all capsule layers. The first among them has **B** number of capsule types. This is the primary capsule layer and the pose matrix of these capsules are learned transformations of outputs of the previous layer. This primary capsule layer is followed by one capsule layer which has **C** number of capsule types with stride 2 and a layer following this containing **D** number of capsule types with stride 1. After this is the final layer of our network which has one capsule type per output class.

Challenges

During this course of our implementation of the models, We encountered multiple issues:

- Sensitivity to hyperparameters : Hinton et.al [8] have not released their official code base for Matrix Capsules. Any open-source implementation [1], [2], we tried and implemented ourselves suffered from severe hyper-parameter sensitivity. The sensitivity affected the convergence and overall accuracy of the model. We suspect this might be due to some underlying design decisions which were not mentioned in their paper.

- Computationally expensive : CapsNet and Matrix Capsules are quite computationally expensive and slow. We were forced to curtail the number of epochs for convergence.

EVALUATION

We used Tesla P-100 GPU cluster from Advance Research Computing at Virginia Tech for our evaluation. We evaluated Matrix capsule networks with EM routing for various dataset shown in table I. These datasets corresponds to small , large and big work load. MNIST is the smallest training work load, SmallNORB and Cifar-10 corresponds to medium training work load. While TinyImageNet corresponds to considerably large training work load. The logs for analysis can be found here

DataSet	Dimensions	Channels	Category	Num Images
Cifar-10	32X32	3	10	60K
SmallNORB	-	4	5	46.6K
TinyImageNet	64X64	3	200	1M
MNIST	28X28	2	10	60K

TABLE I
TABLE TO SHOW DIFFERENT DATASET

We tried our best to train Tiny ImageNet and cifar 10 dataset by varying **A,B,C,D, r** and **optimizer** for Matrix Capsule with EM routing. But for all combinations, we tried we never got a convergence on these datasets. We show one such run in fig 2 and fig 3.

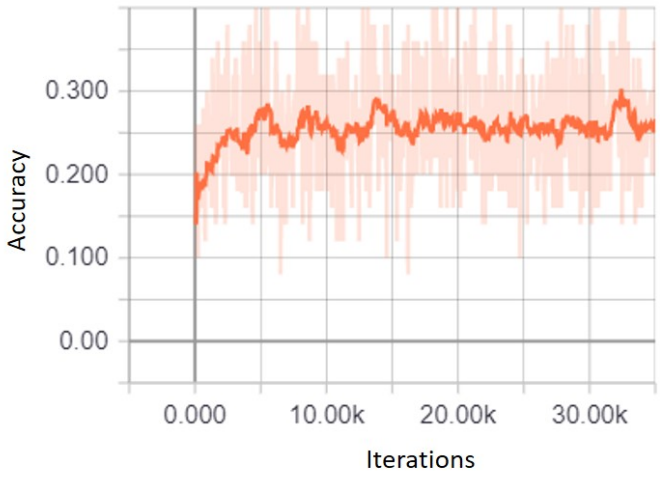

Fig. 2. Accuracy for cifar 10 training

Our next experiment goal is to understand convergence behavior with variation in hyper parameters like **A,B,C,D** and EM step (**r**) for Matrix capsule networks [8].

*Variation in **A***

This experiment correspond to varying **A** which represents the number of channels in the first convolution layer. We fixed other hyperparameters like **B,C,D, optimizer** and **r** to 8,16,16,adam and 2 respectively. We evaluated **A** for three values of 16, 32 and 64 as shown in fig 4. As the capacity of network has increased it converges must faster for **A** with higher values in fig 4. This experiment was conducted with smallNORB dataset.

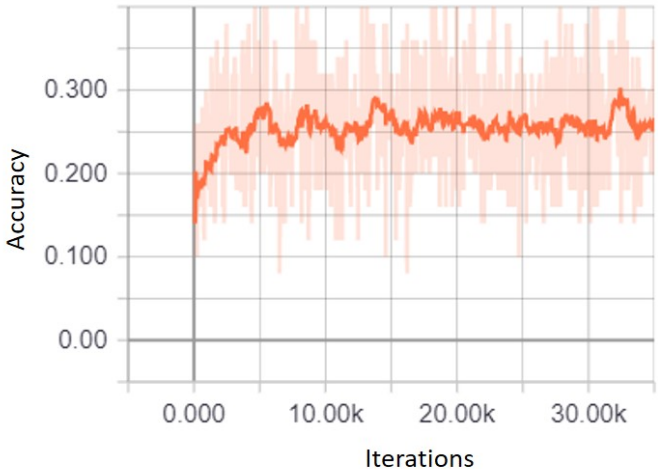

Fig. 3. Accuracy for Tiny ImageNet training

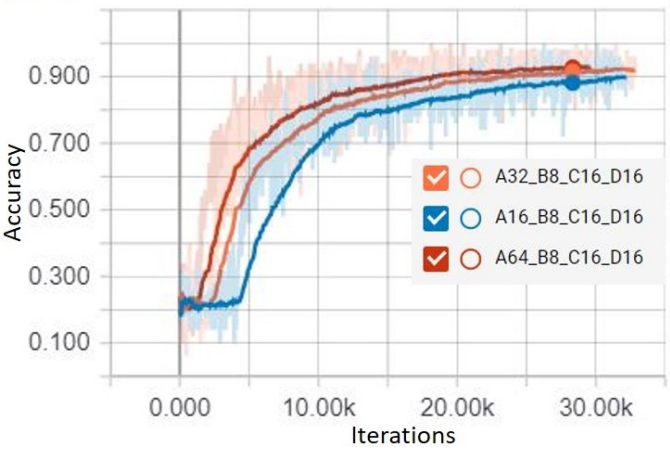

Fig. 4. Varying A for Matrix Capsule Network for smallNORB dataset

*Variation in **B***

This experiment correspond to varying **B** which represents the number of primary capsule layers. We fixed other hyperparameters like **A,C,D, optimizer** and **r** to 32,16,16,adam and 2 respectively. We evaluated **B** for three values of 4, 8 and 32 as shown in fig 5. As the capacity of network has increased it converges must faster for **B** with higher values in fig 5. This experiment was conducted using smallNORB dataset.

We conducted similar experiments using MNIST dataset. We varied **A** for 8,16,32,64 and kept **B,C,D, r** and optimizer constant at 8,16,16, 1 and adam respectively as shown in fig 6. In another experiment, we varied **B** for 8,16,32 and kept **A,C,D, r** and optimizer constant at 8,16,16, 1 and adam respectively as shown in fig 7. In both the experiments, we

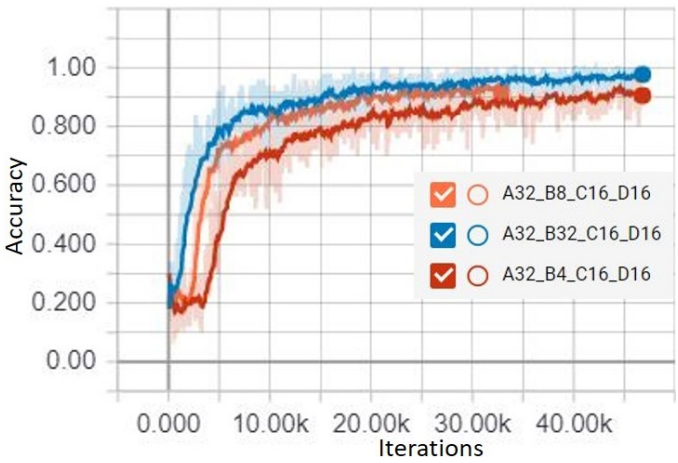

Fig. 5. Varying B for Matrix Capsule Network for smallNORB dataset

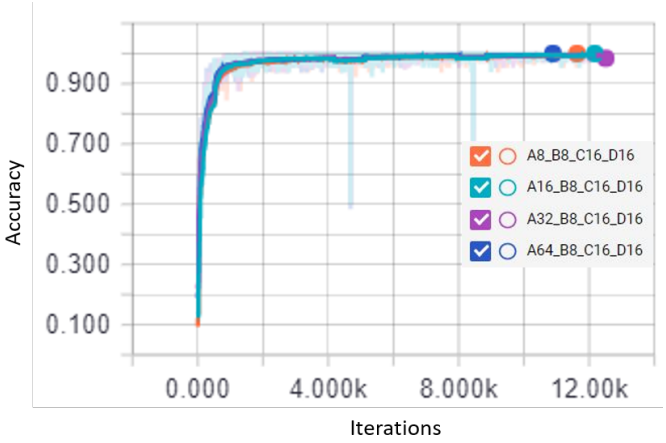

Fig. 6. Varying A for Matrix Capsule Network for MNIST dataset

did not find a very discernible difference in convergence across variation in A and B. We believe that as MNIST dataset is simple with less features, the model is able to learn it quickly in all the settings. Hence, we decided not to continue with further experiments on this dataset. For all future evaluation, we will use smallNORB dataset.

*Variation in **C***

This experiment correspond to varying **C** which represents the number of capsule in convolution layer following primary capsule. We fixed other hyper-parameters like **A,B,D, optimizer** and **r** to 64,8,16,adam and 2 respectively. We evaluated **C** for four values of 4, 8, 16 and 32 as shown in fig 8. As the capacity of network has increased it converges must faster for **C** with higher values in fig 8.

*Variation in **D***

This experiment correspond to varying **D** which represents the number of capsule in the last convolution layer. We fixed other hyper-parameters like **A,B,C, optimizer** and **r** to 32,8,16,adam and 2 respectively. We evaluated **D** for four values of 4, 8, 16 and 32 as shown in fig 9. As the capacity

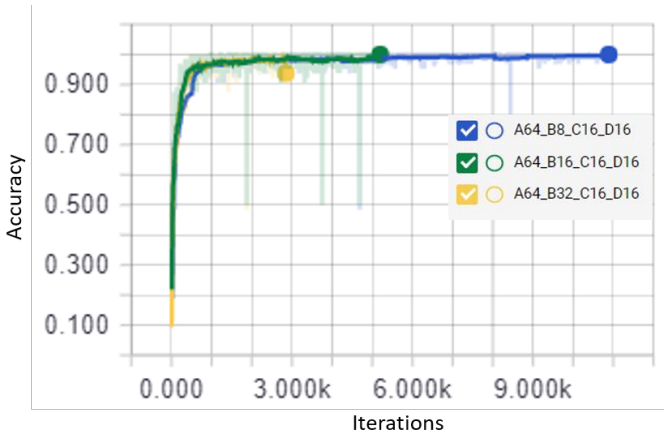

Fig. 7. Varying B for Matrix Capsule Network for MNIST dataset

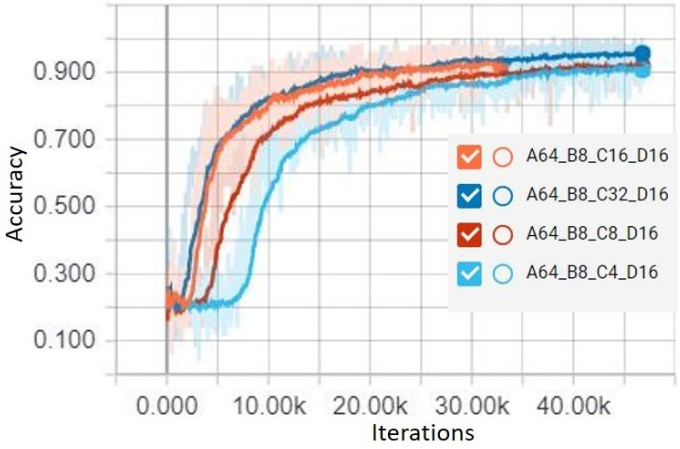

Fig. 8. Varying C for Matrix Capsule Network for smallNORB dataset

of network has increased it converges must faster for **D** with higher values in fig 9.

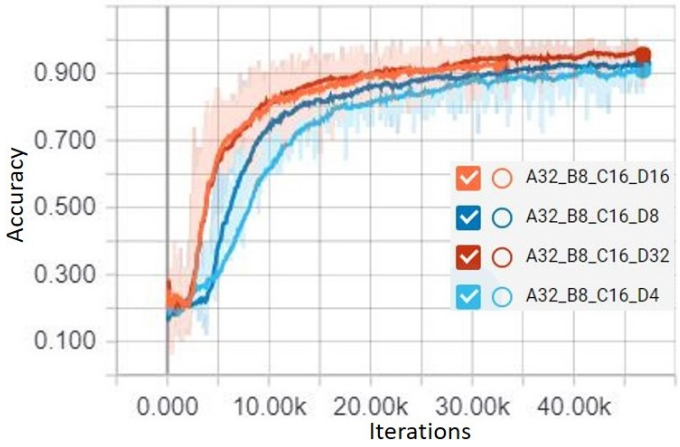

Fig. 9. Varying D for Matrix Capsule Network for smallNORB dataset

*Variation in **r***

The goal of this experiment is to understand convergence behavior with varying EM routing steps. We have fixed other

hyper-parameters like **A,B,C,D** and **optimizers** to 31,8,16,16 and adam respectively for this experiment. We varied r across 4 values of 1,2,3 and 4 as shown in fig 10. We observe that the network only converged for r=2. For r=1, the network has achieved the accuracy of 0.6. While for r=3 and 4, the network is stuck at accuracy of 0.2. We suspect that for r=3,4 the network needs lot more number of epochs to converge and that might the reason for slow progress. We think that r=1, is insufficient number of steps of routing step for converging such complex network, hence the network is stuck at accuracy of 0.6.

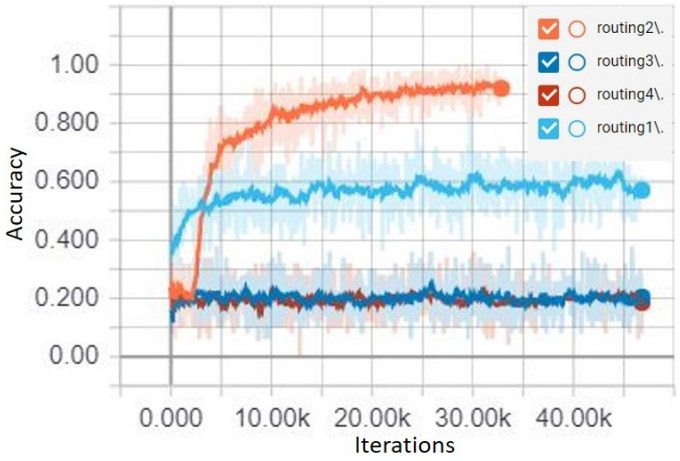

Fig. 10. Routing Step Variation for Matrix Capsule Network for smallNORB dataset

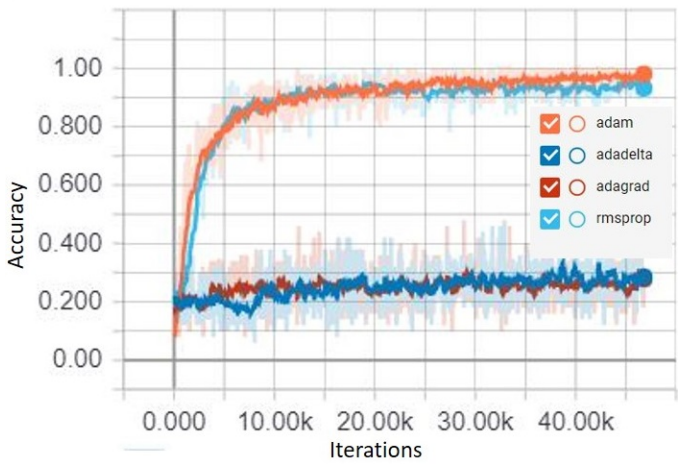

Fig. 11. Optimizers for Capsule Networks with Dynamic Routing for smallNORB dataset

Variation in Optimizers

We performed this evaluation across a cnn model, capsule network with dynamic routing and matrix capsule network with EM routing. The goal is to study rate of convergence across different complexity of network with different optimizers like Adam, Adagrad, Adadelta and RMSprop. We consider CNN to be least complex, capsule net with dynamic routing to be medium complex and matrix capsule net with EM routing as most complex network in our experiment. We are using an exponentially decaying common learning rate for our training. We expect that Adam and RMSprop to converge much faster than Adagrad and Adadelta as general studies show that Adadelta and Adagrad generally converges much slowly for same learning rate.

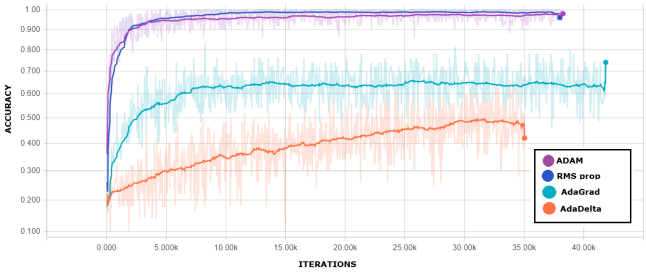

Fig. 12. Optimizers for CNN on smallNORB dataset

For CNN, we observe that RMSProp and Adam has almost similar performance as shown in fig 12. We observe that Agagrad is training much faster than AdaDelta for same learning rate. But both Adagrad and AdaDelta is much slower than Adam and RmsProp.

For capsule networks with dynamic routing [9], we observe that adam and RMSprop performance is almost similar as shown in fig 11 . On the other hand, the network using adagrad and adadelta are training very slow as shown in fig 11.

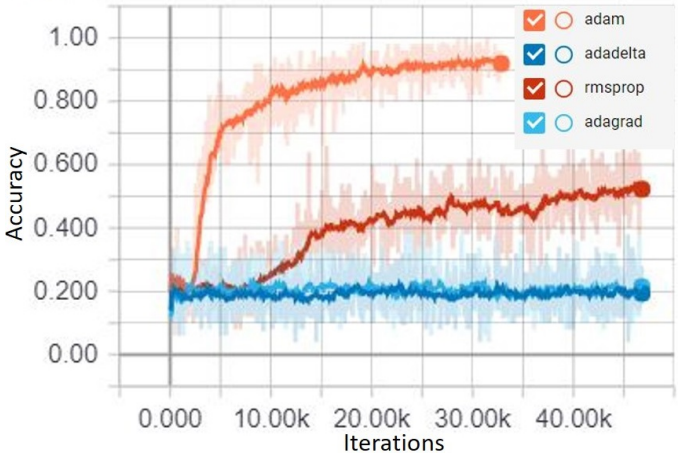

Fig. 13. Optimizers for Matrix Capsule Networks with EM for smallNORB dataset

For matrix capsule network with EM routing, we observe that adam performs best with fastest convergence as shown in fig 13. RMSprop on the other hand, is learning slightly slower and has reached till 52 % accuracy for the same number of iterations. There is no significant training observed for Adagrad and Adadelta for same number of iterations in fig 13.

We observe that with increase in complexity of network, RMSprop does not scale as well as Adam for optimization

for same learning rate. We did not get any good results for Adagrad and Adadelta to perform training of complex network. For less complex networks we find AdaGrad performs better than AdaDelta for same learning rate.

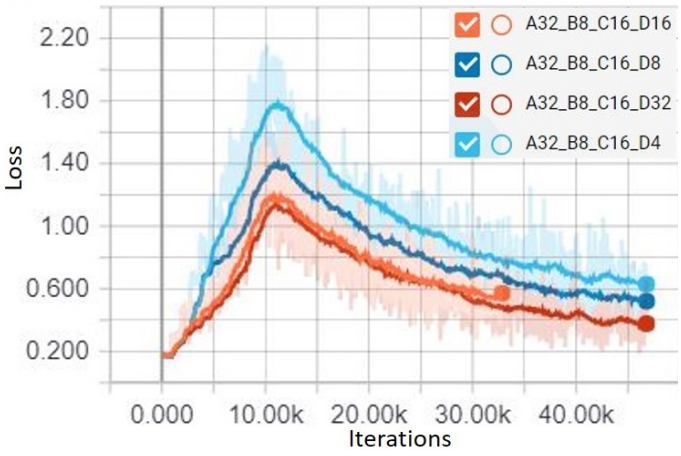

Fig. 14. Loss for Matrix Capsule Network for smallNORB dataset

Loss Behavior

This experiment is designed to understand the behavior of loss function across optimization. We are using the experiment designed for varying **D** in previous section for this evaluation. The behavior of loss function across optimization is shown in fig 14. We observe that initially the loss increases which might be due to convergence of transformation matrix to learn part whole relationship [8]. Later, the loss decreases once the optimizer dominates over the training.

Conclusion

We observed that higher **A,B,C,D** leads to faster training and convergence for matrix capsule networks with EM routing. We found that convergence is very sensitive to number of routing step **r**. For lower routing step, there is not enough iterations for convergence. While higher routing steps increases the complexity of learning leading to need of much more epochs to learn. Our evaluation with optimizers suggested that Adam is the best choice for complex networks, followed by RMSProp. We found that for a given learning rate, Adagrad and Adadelta does not perform as good as Adam and RMSProp. We also observed that the loss of Matrix Capsule Network with EM routing, initially increases due to learning of part whole relationship for transformation matrix and then decreases when the optimizer starts reducing loss.

Future Work

Due to the lack of computation capabilities, we couldnt perform the Grid Search to obtain hyperparameter optimization, which would have been the optimal way to obtain efficient hyperparameters to train bigger datasets like Tiny ImageNet. We would like to perform that in the future. We would also like to provide a framework which can compute the best hyperparameters and optimizers for the Capsule Networks on a given dataset. Also, we would like to extend our work to the ImageNet dataset. We are also interested in investigating the Routing step of the Capsule Networks to see if we can optimize it further.

References

[1] Implementation of matrix capsules in pytorch. https://github.com/shzygmyx/Matrix-Capsules-pytorch.git.

[2] Implementation of matrix capsules in tensorflow. https://github.com/www0wwwjs1/Matrix-Capsules-EM-Tensorflow.

[3] Intuitive explanation of cnns. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets.

[4] Understanding capsule networks. https://medium.com/ai%C2%B3-theory-practice-business/understanding-hintons-capsule-networks-part-i-intuitior

[5] What is capsule network. https://hackernoon.com/what-is-a-capsnet-or-capsule-network-2bfbe48769cc.

[6] What is matrix capsules. https://towardsdatascience.com/demystifying-matrix-capsules-with-em-routing-part-1

[7] Wikipedia convolutional neural network. https://en.wikipedia.org/wiki/Convolutional_neural_network.

[8] Geoffrey Hinton, Nicholas Frosst, and Sara Sabour. Matrix capsules with em routing. 2018.

[9] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in Neural Information Processing Systems*, pages 3859–3869, 2017.

[10] Atefeh Shahroudnejad, Arash Mohammadi, and Konstantinos N Plataniotis. Improved explainability of capsule networks: Relevance path by agreement. *arXiv preprint arXiv:1802.10204*, 2018.

[11] Dilin Wang and Qiang Liu. An optimization view on dynamic routing between capsules. 2018.

[12] Edgar Xi, Selina Bing, and Yang Jin. Capsule network performance on complex data. *arXiv preprint arXiv:1712.03480*, 2017.