HYPERNETWORKS

Vineet(2016CSB1063)
Artificial Neural networks Project Report–CS618

ABSTRACT

This work explores hypernetworks: an approach of using a one network, also known as a hypernetwork, to generate the weights for another network. Hypernetworks provide an abstraction that is similar to what is found in nature: the relationship between a genotype – the hypernetwork – and a phenotype – the main network. The focus of this work is to make hypernetworks useful for convolutional networks. The results also show that hypernetworks applied to convolutional networks still achieve respectable results on MNIST and CIFAR10 as compared to the normal convolution model while requiring fewer learnable parameters.

1 INTRODUCTION

In this work, an approach of using a small network (called a "hypernetwork") to generate the weights for a larger network (called a main network) is considered. The behavior of the main network is the same with any usual neural network: it learns to map some raw inputs to their desired targets; whereas the hypernetwork takes a set of inputs that contain information about the structure of the weights and generates the weight for that layer.

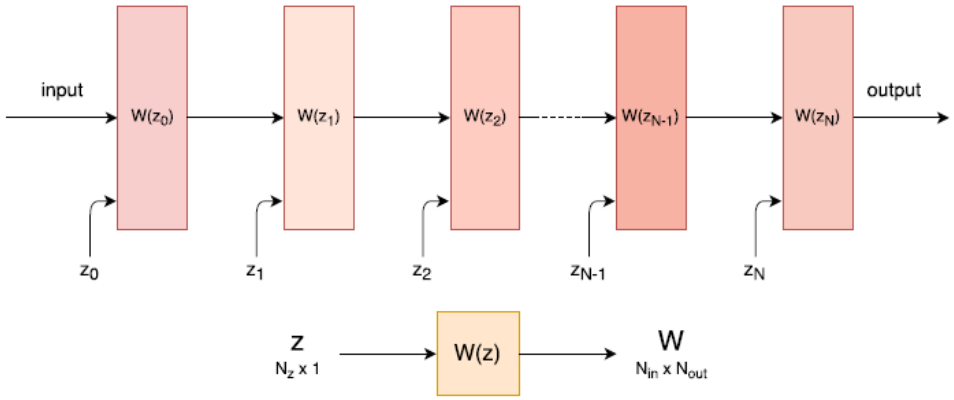


Figure 1: Static Hypernetwork generating weights for the main network

Embedding vector that describes the entire weights of a given layer is given input to the hypernetwork which generates weights for the layer(See Figure 1).Same hypernetwork generates weights for all the layers and each layer is described by its embedding vector. The embedding vectors can be fixed parameters that are also learned during end-to-end training,allowing approximate weight-sharing within a layer and across layers of the main network. The embedding vectors can also be generated dynamically by dynamic hypernetwork, allowing the weights of a recurrent network to change over timesteps and also adapt to the input sequence.
In this work, the behaviour of static hypernetwork is investigated by performing extensive experiments on MNIST and CIFAR-10.

2 RELATED WORK

The work is inspired by methods in evolutionary computing, where it is difficult to directly operate in large search spaces consisting of millions of weight parameters. A more efficient method is to evolve a smaller network to generate the structure of weights for a larger network, so that the search is constrained within the much smaller weight space. An instance of this approach is the work on the HyperNEAT framework (Stanley et al. (2009)). In the HyperNEAT framework, Compositional Pattern-Producing Networks (CPPNs) are evolved to define the weight structure of much larger main network. Most reported results using these methods, however, are in small scales, perhaps because they are both slow to train and require heuristics to be efficient. The main difference between hypernetwork and HyperNEAT is that hypernetworks are trained end-to-end with gradient descent together with the main network, and therefore are more efficient. Even before the work on HyperNEAT ,Schmidhuber (1992) has suggested the concept of fast weights in which one network can produce context-dependent weight changes for a second network. Small scale experiments were conducted to demonstrate fast weights for feed forward networks at the time, but perhaps due to the lack of modern computational tools, the recurrent network version was mentioned mainly as a thought experiment Schmidhuber (1992). A subsequent work demonstrated practical applications of fast weights Gomez & Schmidhuber (2005), where a generator network is learnt through evolution to solve an artificial control problem.
The focus of this work is to generate weights for practical architectures, such as convolutional networks and recurrent networks by taking layer embedding vectors as inputs. However, the hypernetworks can also be utilized to generate weights for a fully connected network by taking coordinate information as inputs similar to DPPNs.

3 METHODOLOGY

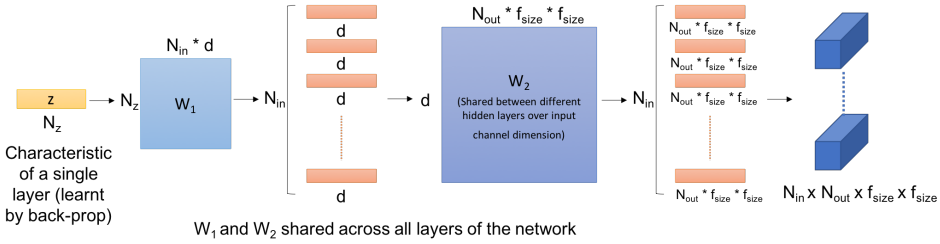
In a typical deep convolutional network, the majority of model parameters are in the kernels of convolutional layers. Each kernel contain $N_{in} \times N_{out}$ filters and each filter has dimensions $fsize \times fsize$. Let's suppose that these parameters are stored in a matrix $K_j \epsilon R^{N_{in} f_{size} out f_{size}}$ for each layer $j = 1, .., D$, where D is the depth of the main convolutional network. For each layer j, the hypernetwork receives a layer embedding $z^j \epsilon R^{Nz}$ as input and predicts K^j , which can be generally written as follows:

$$K_j = g(z^j), \forall j = 1, ..., D$$

We note that this matrix K^j can be broken down as N_{in} slices of a smaller matrix with dimensions $f_{size} \times N_{out} f_{size}$, each slice of the kernel is denoted as$K_i^j \epsilon R^{f_{size} \times N_{out} f_{size}}$. Therefore, the hypernetwork is a two-layer linear network. The first layer of the hypernetwork takes the input vector z^j and linearly projects it into the N_{in} inputs, with N_{in} different matrices$W_i \epsilon R^{d \times N_z}$ and bias vectors $B_i \epsilon R^d$, where d is the size of the hidden layer in the hypernetwork. For simplicity, fix d to be equal to N_z although they can be different. The final layer of the hypernetwork is a linear operation which takes an input vector a_i of size d and linearly projects that into K_i using a common tensor $W_{out} \epsilon R^{f_{size} \times N_{out} f_{size} \times d}$ and bias matrix $B_{out} \epsilon R^{f_{size} \times N_{out} f_{size}}$. The final kernel K_j will be a concatenation of every K_i^j . Thus $g(z^j)$ can be written as follows

$$\begin{aligned}
a_i^j &= W_i z^j + B_i, && \forall i = 1, .., N_{in}, \forall j = 1, ..., D \\
K_i^j &= \langle W_{out}, a_i^j \rangle^{1} + B_{out}, && \forall i = 1, .., N_{in}, \forall j = 1, ..., D \\
K^j &= \begin{pmatrix} K_1^j & K_2^j & ... & K_i^j & ... & K_{N_{in}}^j \end{pmatrix}, && \forall j = 1, ..., D
\end{aligned}$$

The above approach can be visualized in the figure 3
The above formulation assumes that the network architecture consists of kernels with same dimensions. In practice, deep convolutional network architectures consists of kernels of varying dimensions. Typically, in many designs, the kernel dimensions are integer multiples of a basic size.To modify our approach to work with this architecture, we have our hypernetwork generate kernels for this basic size of 16, and if we require a larger kernel for a certain layer, we will concatenate multiple basic kernels together to form the larger kernel. If the kernel consist of 32 x 32 channels then total 4 embedding for that layer will be used each will generate 16 x 16 kernel.



Block1	Conv 3x3,16
	Conv 3x3,16
	MaxPool (2,2)
	Dropout 0.4
Block2	Conv 3x3,32
	Conv 3x3,32
	MaxPool (2,2)
	Dropout 0.4
Block3	Conv 3x3,64
	Conv 3x3,64
	MaxPool (2,2)
	Dropout 0.4
	Dense(128)
	Dropout
	Dense(10)

Table 1: Model architecture for CIFAR-10

3.1 MNIST ARCHITECTURE

The main convolutional network is a small two layer network and the hypernetwork is used to generate the kernel for the second layer $(7 \times 7 \times 16 \times 16)$, which contains the bulk of the trainable parameters in the system. Training is done on different embedding size.
The network is trained with a 55000 / 5000 / 10000 split for the training, validation and test sets and use the 5000 validation samples for early stopping, and train the network using Adam Kingma & Ba (2014) with a learning rate as hyperparameter and with decay of 0.999(multiplied every step) on mini-batches of size 1000. To decrease overfitting, the MNIST training images are padded to 30x30 pixels and random crop to 28x28.

3.2 CIFAR-10 ARCHITECTURE

The main convolutional network is based on VGG network (Simonyan & Zisserman (2014)). Three main models are trained. In each model one VGG block is added. The final main convolutional network consists of three VGG blocks and final two dense layers as described in Table 3.2. The three models are named as: **VGG-Block1**,**VGG-Block2** and **VGG-Block3** where block2 consists of two blocks and block3 consists of three blocks along with the last two dense layer.The hypernetwork generate kernel of 16x16 in all models.
Both the normal networks and the hypernetwork version are trained using a 45000 / 5000 /10000 split for training, validation, and test set. Training is done with a mini-batch size of 128 using Adam for the hypernetwork version with learning rate decay. To decrease over fitting, data augmentation is applied by padding training images to 36x36 pixels and random crop to 32x32, and dropout at last dense as hyperparameter.

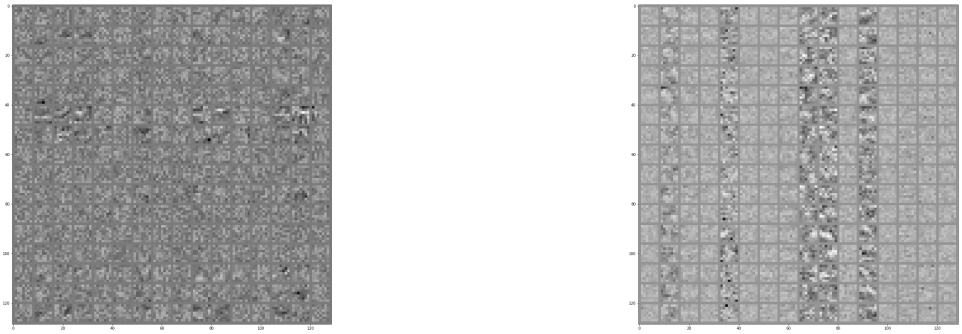


Figure 2: Kernels learned by a ConvNet to classify MNIST digits (left). Kernels learned by a hypernetwork generating weights for the ConvNet (right).

4 RESULTS AND DISCUSSION

4.1 MNIST

The results on MNIST are shown in Table 2. For this task, the hypernetwork achieved a test accuracy of 98.24%, comparable to the 98.67% for the conventional method. In this example, a kernel consisting of 12,560 weights is represented by an embedding vector of only 5 parameters, generated by a hypernetwork that has 5,189 parameters.Now the question is whether we can also train a deep convolutional network, using a single hypernetwork generating a set of weights for each layer, on a dataset more challenging than MNIST.

Model	Best Test Accuracy	Conv Parameters
ConvModel	98.67%	12,560
Hyper4_4	98.12%	4,244
Hyper4_5	98.67%	5,108
Hyper4_6	98.75%	5,972
Hyper5_5	**98.79**%	5,189
Hyper6_6	98.14%	6,166
Hyper4_7	98.61%	6,836
Hyper4_8	98.55%	7,700
Hyper7_7	98.24%	7,175

Table 2: Results on MNIST. Here Conv parameters does not consists of first convolution layer. HyperA_B means N_z(embedding dimension) is A and D(embedding projection dimension) is B.

4.2 CIFAR-10

The results on CIFAR-10 are shown in Table 3, 4 and 5 for VGG-Block1 , VGG-Block2 and VGG-Block3. The accuracy for ConvModel increases from Block 1 to Block 3 but the increase in number of parameter is very high . In VGG-Block 1 and VG-Block 2 the best accuracy achieved by hypernetworks are 64.87% and 67.02% whereas the increase in parameter is less compared to the ConvModel. There is decrease in accuracy for Hyper4_4 and Hyper6_6 from VGG Block1 to VGG Block2. One reason for this reduction in accuracy is because different layers of a deep network is trained to extract different levels of features, and require different kinds of filters to perform optimally. The hypernetwork enforces some commonality between every layer.While the network is no longer able to learn the optimal set of filters for each layer, it will learn the best set of filters given the constraints, and the resulting number of model parameters is drastically reduced.

Model	Best Test Accuracy	Conv Parameters
ConvModel	**71.05**%	2,320
Hyper4_4	**64.87**%	1,044
Hyper6_6	63.82%	1,686
Hyper4_5	63.11%	1,268
Hyper4_6	58.06%	1,492

Table 3: Results of VGG-Block1 model on CIFAR. Here Conv parameters does not consists of first convolution layer. HyperA_B means N_z(embedding dimension) is A and D(embedding projection dimension) is B.

Model	Best Test Accuracy	Conv Parameters
ConvModel	**74.76**%	16,208
Hyper4_4	58.82%	1,056
Hyper6_6	56.66%	1,722
Hyper8_8	59.77%	2,504
Hyper16_16	**67.02**%	6,912

Table 4: Results of VGG-Block2 model on CIFAR. Here Conv parameters does not consists of first convolution layer. HyperA_B means N_z(embedding dimension) is A and D(embedding projection dimension) is B.

Model	Best Test Accuracy	Conv Parameters
ConvModel	**75.06**%	71,632
Hyper8_8	47.79%	2,696
Hyper8_16	56.66%	7,196
Hyper16_16	56.66%	7,196
Hyper32_32	57.98%	22,640

Table 5: Results of VGG-Block3 model on CIFAR. Here Conv parameters does not consists of first convolution layer. HyperA_B means N_z(embedding dimension) is A and D(embedding projection dimension) is B.

5 CONCLUSIONS

In this work, many experiments on static hypernetworks were performed.In most of the cases the method works well while using fewer parameters. Although many hyperparameter settings were considered while training, more experiments/evaluations are required to compare the hypernetworks on CIFAR-10.
In conclusion, we can say that an interesting new method for training neural networks is used, i.e., a hypernetwork is used to generate the model parameters of the main network. It demonstrated that the total number of model parameters could be smaller while achieving competitive results on the image classification task.

REFERENCES

Faustino Gomez and Jürgen Schmidhuber. Evolving modular fast-weight networks for control. In *International Conference on Artificial Neural Networks*, pp. 383–389. Springer, 2005.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Kenneth O Stanley, David B D'Ambrosio, and Jason Gauci. A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, 15(2):185–212, 2009.

A MNIST

Model	Learning rate	Validation Accuracy	Testing Accuracy	Parameters of 2nd Kernel	Total Parameters
1	0.003	98.98%	98.65%	12560	21,210
	0.004	98.88%	**98.67**%	12560	21,210
	0.005	98.64%	98.46%	12560	21,210

Table 6: Hypertuning results for convmodel on MNIST

Model	Emb Dim	Proj Dim	Learning rate	Validation Accuracy	Testing Accuracy	2nd Kernel Parameters
1	4	4	0.003	98.56%	**98.12**%	4244
			0.004	98.58%	98.06%	
			0.005	98.68%	97.62%	
2	4	5	0.003	98.64%	**98.67**%	5108
			0.004	98.72%	98.33%	
			0.005	98.76%	98.36%	
3	4	6	0.003	98.98%	**98.75**%	5972
			0.004	98.42%	97.97%	
			0.005	98.96%	98.61%	
4	4	7	0.003	98.88%	**98.61**%	6836
			0.004	98.38%	98.26%	
			0.005	97.26%	96.80%	
5	4	8	0.003	98.14%	97.52%	7700
			0.004	98.42%	98.17%	
			0.005	99.08%	**98.55**%	
6	5	5	0.003	98.64%	98.22%	5189
			0.004	99.02%	**98.79**%	
			0.005	98.70%	98.34%	
7	6	6	0.003	98.30%	**98.14**%	6166
			0.004	98.62%	98.11%	
			0.005	98.54%	98.05%	
8	7	7	0.003	98.70%	**98.24**%	7175
			0.004	98.58%	98.05%	
			0.005	98.54%	97.99%	

Table 7: Hypertuning results for hypernetwork on MNIST