Neural Architecture Search With Reinforcement Learning

Barret Zoph, Quoc V. Le Google Brain

ERKUT AKDAĞ

2194538

11.05.2017

Middle East Technical University

CENG793 Advanced Deep Learning

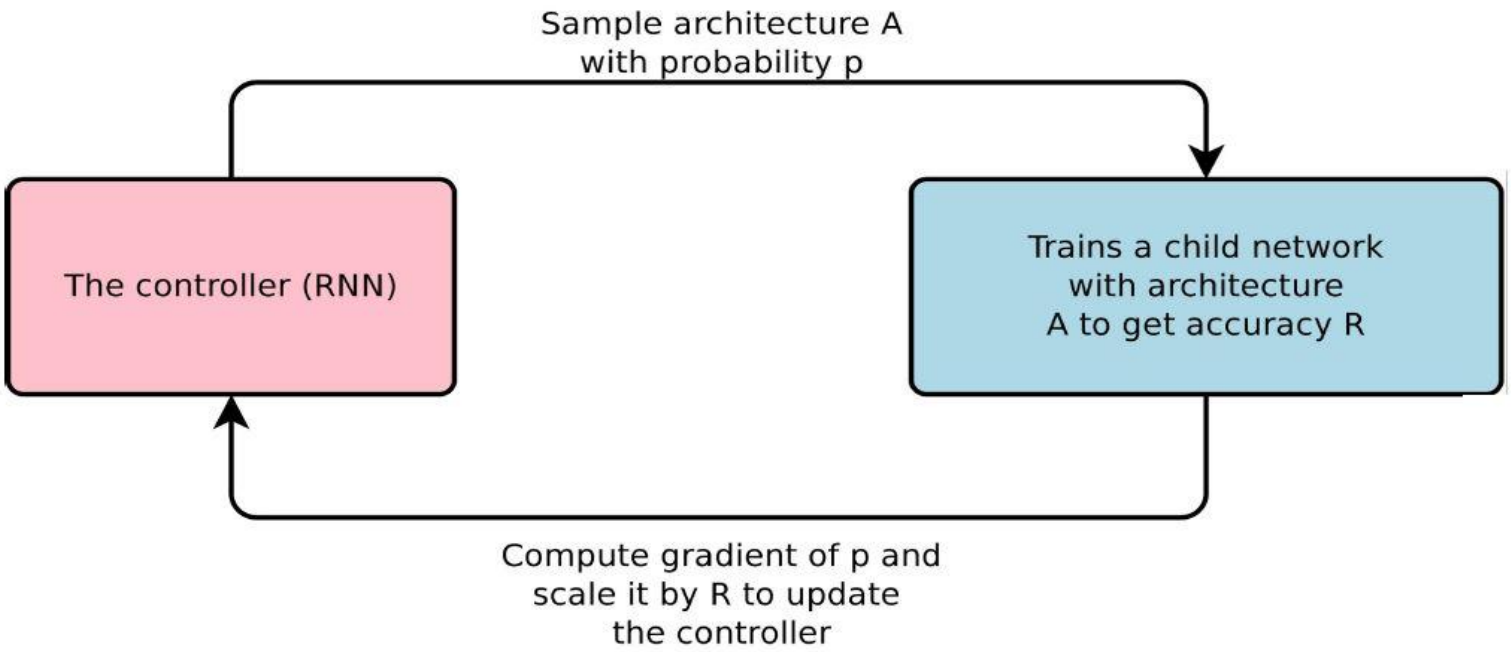
OUTLINE

- Introduction
- Related Work
- Methods
- Experiments and Results
- Conclusion

Introduction

- Neural networks are powerful and flexible models.
- The last few years have been much success of deep neural networks in many challenging applications but designing architectures still requires a lot of expert knowledge and takes ample time.
- Despite their success neural networks are still hard to design.
- In this paper, recurrent network is used in order to generate the model descriptions of neural networks.
- Train this RNN with reinforcement learning in order to maximize the expected accuracy of the generated architectures (on a validation set).

Introduction



- This paper represents the "Neural Architecture Search", a gradient-based method for finding good architectures.
- This work is based on the observation that the structure and connectivity of neural network can be typically specified by a variable-length string.

Introduction

- It is possible to use a recurrent network –the controller- to generate such string.
- Training the network specified by the string- the "child network"- on the real data will result in an accuracy on a validation set. Using this accuracy as the reward signal, we can compute the policy gradient to update the controller.
- As a result, in the next iteration, the controller will give higher probabilities to architectures. In other words, the controller will learn to improve its search over time.

Introduction

- On CIFAR-10
  - novel ConvNet model is found.
  - Better than previous most human-invented architecture.
  - 0.09 percent better error rate
  - 0.05 faster than previous state of art models
- On Penn Treebank
  - Novel recurrent cell is found.
  - Better than previous RNN&LSTM.
  - 3.6 perplexity better than previous state of art models.

Related Work

- **Hyperparameter optimization** is an important research and only search models from a fixed-length space. These methods often work better if they are supplied with a good initial model.
- **Modern neuro-evolution** algorithms much more flexible for composing novel models, but they are usually less practical at a large scale.
- The idea of learning to learn or meta-learning. Neural network is used to learn the gradient descent updates for another network and the idea of using reinforcement learning to find update policies for another network.

Related Work

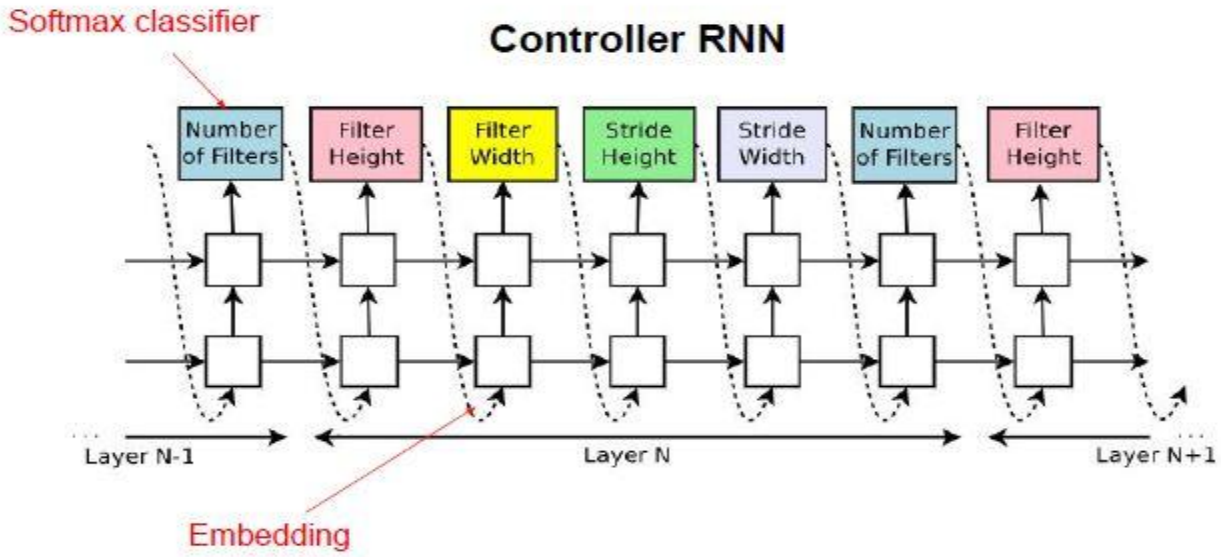
- The controller in Neural Architecture Search is auto-regressive.
- It predicts hyperparameters one a time, conditioned on previous predictions. This idea is borrowed from the **decoder in end-to-end sequence to sequence learning**.
- Unlike sequence to sequence learning, this model optimizes the accuracy (non-differantiable metric) of the child network.
- It is similar to the work on BLEU optimization in Neural Machine Translation. Unlike these approaches, this method learns directly from the reward signal without any supervised bootstrapping.

Methods

- Using recurrent network
  - To generate convolutional architecture
- How the recurrent network can be trained with a policy gradient method
  - to maximize the expected accuracy of the sampled architectures.
- Several improvements of core approach
  - forming skip connections to increase model complexity
  - using a parameter server approach to speed up training.
- Generating recurrent cell architectures

Methods

- **Controller Recurrent Neural Network**



- In Neural Architecture Search, controller is used to generate architectural hyperparameters of neural networks.
- Controller predicts filter height, filter width, stride height, stride width, and number of filters for one layer and repeats. Every prediction is carried out by softmax classifier and then fed into the next time step as input.

Methods

- **Controller Recurrent Neural Network**
- In the experiments, the process of generating an architecture stops if the number of layers exceeds a certain value. This value follows a schedule where they increase it as training progresses.
- After the controller RNN finishes generating an architecture, a neural network with this architecture is built and trained. At convergence, the accuracy of the network on a held-out validation set is recorded.
- The parameters of the controller RNN, θ_c ,are then optimized in order to maximize the expected validation accuracy. Policy gradient method is used to update parameters θ_c for the generating better architectures over time by controller RNN.

Methods

- **Training with REINFORCE**
- The list of tokens that the controller predicts can be viewed as a list of actions to design an architecture for a child network.
- At convergence, this child network will achieve an accuracy R on a held-out dataset.
- This accuracy R as the reward signal and use reinforcement learning to train the controller. Maximize the expected reward in order to find the optimal architecture.

Methods

- **Training with REINFORCE**

Parameters of Controller RNN

Accuracy of architecture on held-out dataset

$$J(\theta_c) = E_{P(a_{1:T};\theta_c)}[R]$$

Architecture predicted by the controller RNN viewed as a sequence of actions

$$\nabla_{\theta_c} J(\theta_c) = \sum_{t=1}^{T} E_{P(a_{1:T};\theta_c)} \left[\nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c) R \right]$$

Number of models in minibatch

$$\frac{1}{m} \sum_{k=1}^{m} \sum_{t=1}^{T} \nabla_{\theta_c} \log P(a_t | a_{(t-1):1}; \theta_c) R_k$$

- Policy gradient method is used iteratively update θ_c (R is non-differantiable)
- Unbiased estimate for gradient, very high variance.

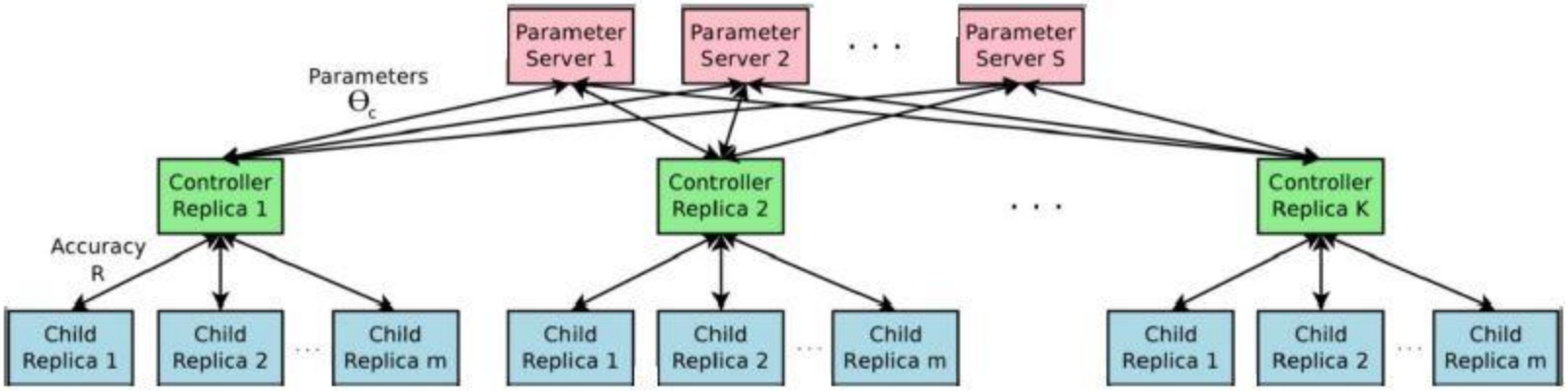
Methods

- **Training with REINFORCE**
- In order to decrease the variance of this estimate baseline function can be used in equation.
- Baseline function "b" is exponential moving average of the previous architecture accuracies.

$$\frac{1}{m}\sum_{k=1}^{m}\sum_{t=1}^{T}\nabla_{\theta_c}\log P(a_t|a_{(t-1):1};\theta_c)(R_k-b)$$

Methods

- **Accelerate Training with Parallelism and Asynchronous Updates**



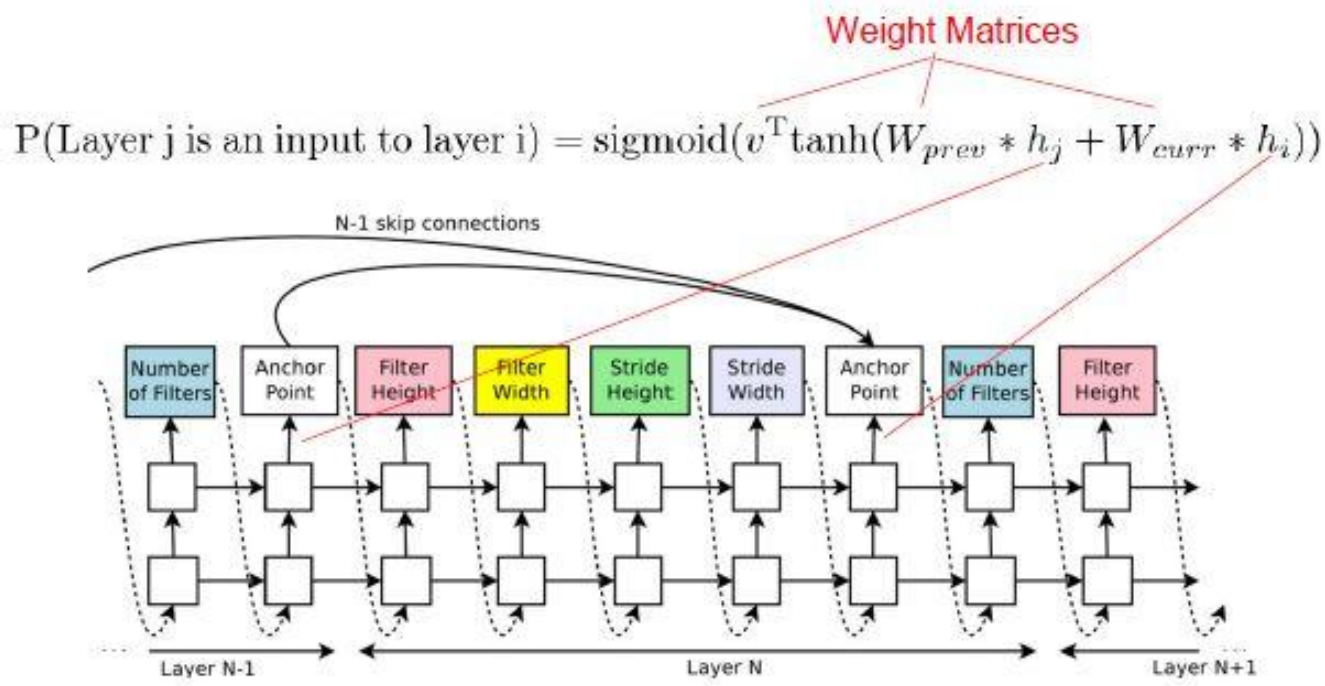
- As training a child network can take hours, distributed training and asynchronous parameter updates in order to speed up the learning process of the controller.
- Parameter-server scheme where we have a parameter server of S shards, that store the shared parameters for K controller replicas. Each controller replica samples m different child architectures that are trained in parallel.

Methods

- **Accelerate Training with Parallelism and Asynchronous Updates**
- The controller then collects gradients according to the results of that minibatch of m architectures at convergence and sends them to the parameter server in order to update the weights across all controller replicas.
- In this implementation, convergence of each child network is reached when its training exceeds a certain number of epochs.

Methods

- **Increase Architecture Complexity with skip connections and other layer types**



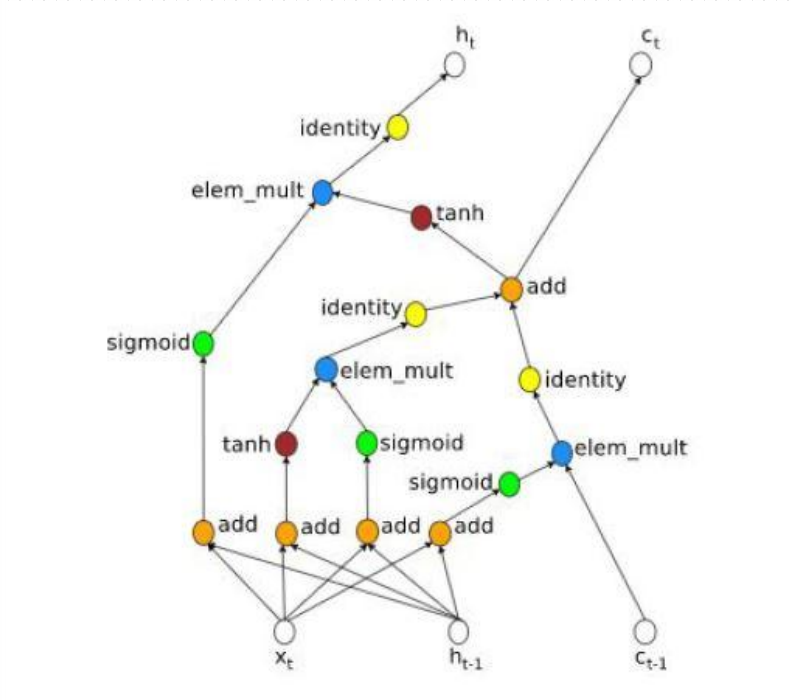
- Set-selection type attention is used. At layer N, anchor point is added.

Methods

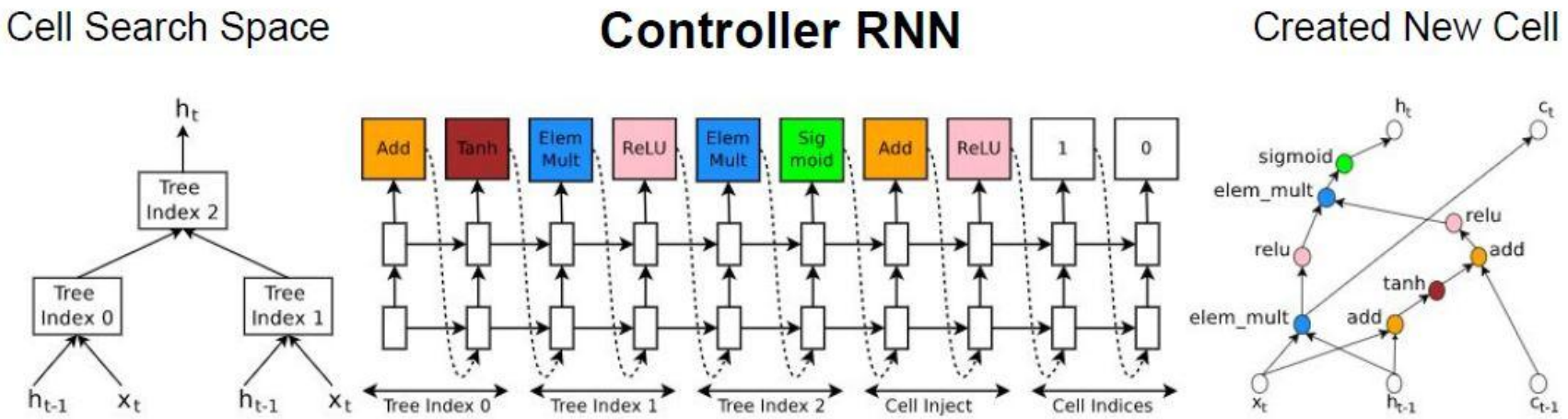
- Skip connections can cause ”compilation failures” where one layer is not compatible with another layer, or one layer may not have any input or output.
- To overcome these issues, three simple techniques are applied.
- First, if a layer is not connected to any input layer then the image is used as the input layer.
- Second, at the final layer they take all layer outputs that have not been connected and concatenate them before sending this final hidden state to the classifier.
- Lastly, if input layers to be concatenated have different sizes, we pad the small layers with zeros so that the concatenated layers have the same sizes.

Methods

- **Recurrent Cell Architecture Generate**
- Created a search space for search over RNN cells like the LSTM or GRU
- Based our search space off the LSTM cell in that they have a recurrent state and cell



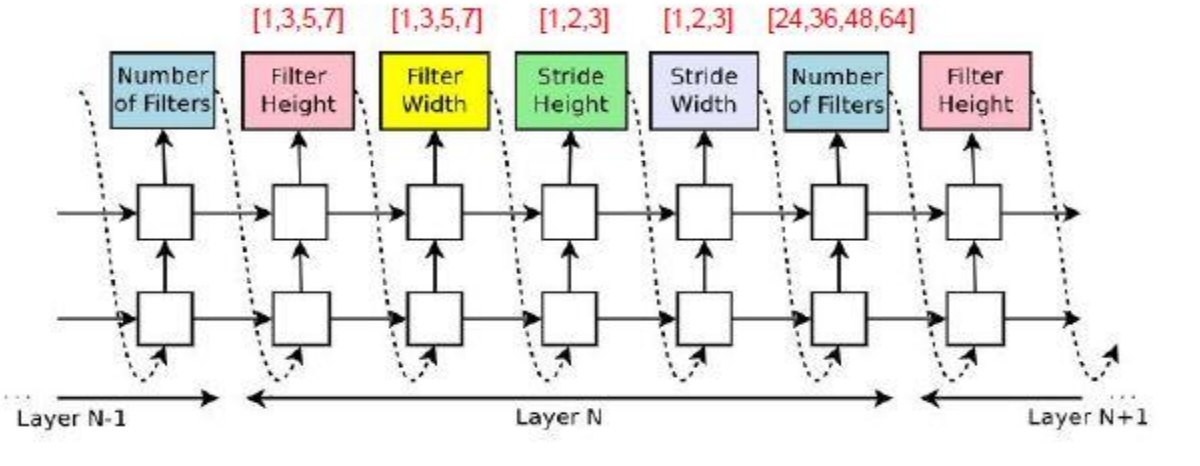
Methods



- The controller predicts Add and $Tanh$ for tree index 0, this means we need to compute $a_0 = \tanh(W_1 * x_t + W_2 * h_{t-1})$.
- The controller predicts $ElemMult$ and $ReLU$ for tree index 1, this means we need to compute $a_1 = \mathrm{ReLU}\big((W_3 * x_t) \odot (W_4 * h_{t-1})\big)$.
- The controller predicts 0 for the second element of the "Cell Index", Add and $ReLU$ for elements in "Cell Inject", which means we need to compute $a_0^{new} = \mathrm{ReLU}(a_0 + c_{t-1})$. Notice that we don't have any learnable parameters for the internal nodes of the tree.
- The controller predicts $ElemMult$ and $Sigmoid$ for tree index 2, this means we need to compute $a_2 = \mathrm{sigmoid}(a_0^{new} \odot a_1)$. Since the maximum index in the tree is 2, h_t is set to a_2.
- The controller RNN predicts 1 for the first element of the "Cell Index", this means that we should set c_t to the output of the tree at index 1 before the activation, i.e., $c_t = (W_3 * x_t) \odot (W_4 * h_{t-1})$.

Experiments & Results

- **Learning Convolutional Architecture for CIFAR-10**
  - Preprocessing and augmentation procedures
- **Search Space**
  - Convolutional architecture
  - Rectified linear units
  - Batch normalization
  - Skip connection btw layers



Experiments & Results

- **Training Details**
  - Two-layer LSTM
  - 35 hidden units
  - ADAM optimizer
  - Learning rate: 0.0006
  - The weight init: -0.08 +0.08 uniform
  - Server shards(S): 20
  - Controller replicas(K): 100
  - Child replicas(m): 8
  - Totally 800 networks trained over 800 GPU's

Experiments & Results

- **Training Details**
  - 50 epochs
  - The reward used for updating the controller : max validation accuracy of the last 5 epoch
  - Validation set: 5000 examples (randomly chosen)
  - Training set: 45000 examples
  - Momentum optimizer: 0.1
  - Weight decay: 1e-4
  - Momentum: 0.9
  - Nesterov Momentum

Experiments & Results

Model	Depth	Parameters	Error rate (%)
Network in Network (Lin et al., 2013)	-	-	8.81
All-CNN (Springenberg et al., 2014)	-	-	7.25
Deeply Supervised Net (Lee et al., 2015)	-	-	7.97
Highway Network (Srivastava et al., 2015)	-	-	7.72
Scalable Bayesian Optimization (Snoek et al., 2015)	-	-	6.37
FractalNet (Larsson et al., 2016)	21	38.6M	5.22
with Dropout/Drop-path	21	38.6M	4.60
ResNet (He et al., 2016a)	110	1.7M	6.61
ResNet (reported by Huang et al. (2016c))	110	1.7M	6.41
ResNet with Stochastic Depth (Huang et al., 2016c)	110	1.7M	5.23
	1202	10.2M	4.91
Wide ResNet (Zagoruyko & Komodakis, 2016)	16	11.0M	4.81
	28	36.5M	4.17
ResNet (pre-activation) (He et al., 2016b)	164	1.7M	5.46
	1001	10.2M	4.62
DenseNet ($L = 40, k = 12$) Huang et al. (2016a)	40	1.0M	5.24
DenseNet($L = 100, k = 12$) Huang et al. (2016a)	100	7.0M	4.10
DenseNet ($L = 100, k = 24$) Huang et al. (2016a)	100	27.2M	3.74
DenseNet-BC ($L = 100, k = 40$) Huang et al. (2016b)	190	25.6M	3.46
Neural Architecture Search v1 no stride or pooling	15	4.2M	5.50
Neural Architecture Search v2 predicting strides	20	2.5M	6.01
Neural Architecture Search v3 max pooling	39	7.1M	4.47
Neural Architecture Search v3 max pooling + more filters	39	37.4M	3.65

5% faster

Experiments & Results

- **Learning Recurrent Cells for Penn Treebank**
  - Regularization methods are needed to avoid over fitting
  - Embedding dropout and recurrent dropout

- **Search Space**
  - Combination method and activation function for each node in tree
  - Combination method [add, elem_mult]
  - Activation method[identity, tanh, sigmoid, relu]
  - Number of input pairs to RNN cell: base number: 8

Experiments & Results

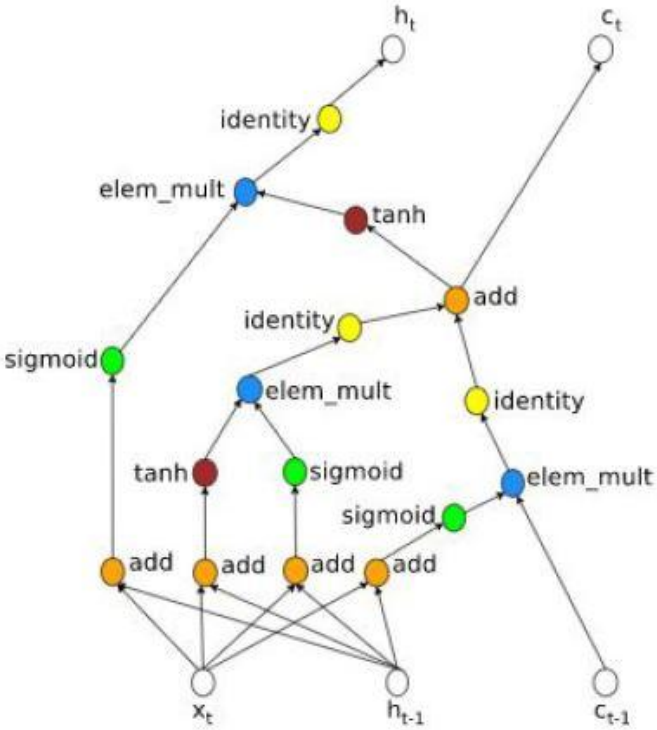
- **Training Details**
  - Learning rate: 0.0005
  - Server shards(S): 20
  - Controller replicas(K): 400
  - Child replicas(m): 1
  - Totally 400 networks trained over 400 GPU's
  - 35 epochs
  - Child model has 2 layers

Experiments & Results

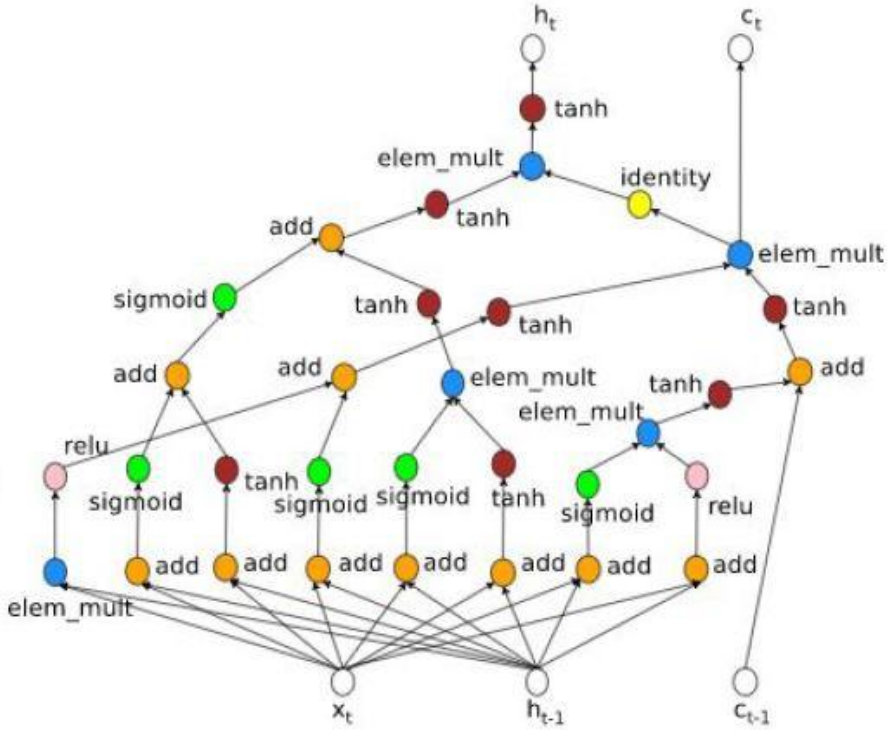
Model	Parameters	Test Perplexity
Mikolov & Zweig (2012) - KN-5	2M‡	141.2
Mikolov & Zweig (2012) - KN5 + cache	2M‡	125.7
Mikolov & Zweig (2012) - RNN	6M‡	124.7
Mikolov & Zweig (2012) - RNN-LDA	7M‡	113.7
Mikolov & Zweig (2012) - RNN-LDA + KN-5 + cache	9M‡	92.0
Pascanu et al. (2013) - Deep RNN	6M	107.5
Cheng et al. (2014) - Sum-Prod Net	5M‡	100.0
Zaremba et al. (2014) - LSTM (medium)	20M	82.7
Zaremba et al. (2014) - LSTM (large)	66M	78.4
Gal (2015) - Variational LSTM (medium, untied)	20M	79.7
Gal (2015) - Variational LSTM (medium, untied, MC)	20M	78.6
Gal (2015) - Variational LSTM (large, untied)	66M	75.2
Gal (2015) - Variational LSTM (large, untied, MC)	66M	73.4
Kim et al. (2015) - CharCNN	19M	78.9
Press & Wolf (2016) - Variational LSTM, shared embeddings	51M	73.2
Merity et al. (2016) - Zoneout + Variational LSTM (medium)	20M	80.6
Merity et al. (2016) - Pointer Sentinel-LSTM (medium)	21M	70.9
Inan et al. (2016) - VD-LSTM + REAL (large)	51M	68.5
Zilly et al. (2016) - Variational RHN, shared embeddings	24M	66.0
Neural Architecture Search with base 8	32M	67.9
Neural Architecture Search with base 8 and shared embeddings	25M	64.0
Neural Architecture Search with base 8 and shared embeddings	54M	62.4

2x as fast

Experiments & Results



LSTM Cell



Neural Architecture Search (NAS) Cell

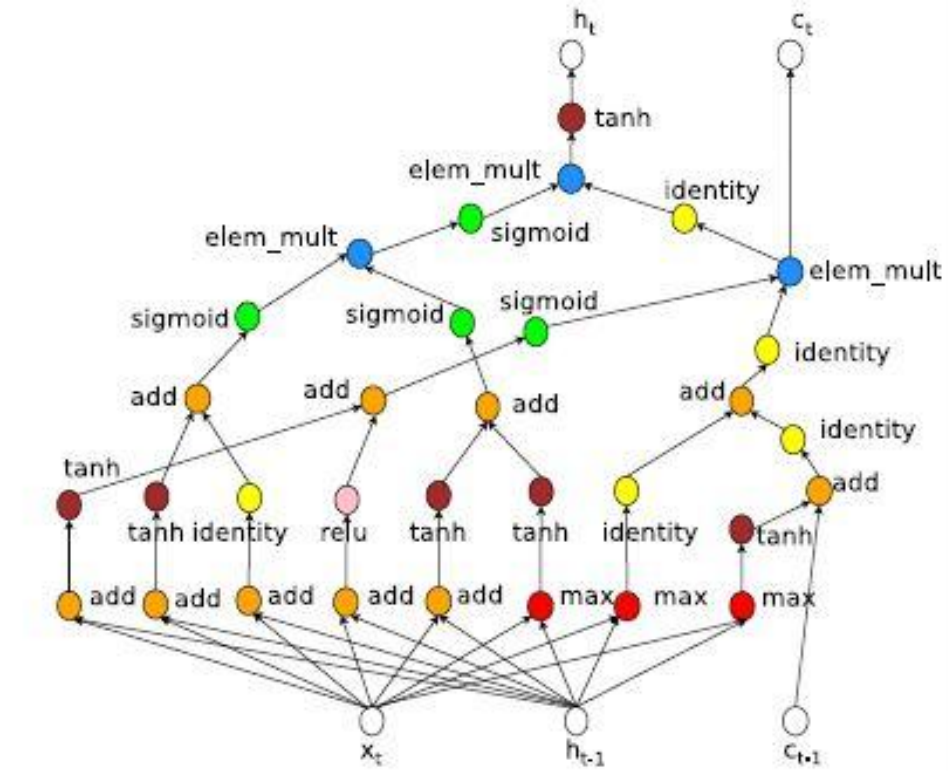
Experiments & Results

- **Transfer Learning on Character Level Language Modeling**

RNN Cell Type	Parameters	Test Bits Per Character
Ha et al. (2016) - Layer Norm HyperLSTM	4.92M	1.250
Ha et al. (2016) - Layer Norm HyperLSTM Large Embeddings	5.06M	1.233
Ha et al. (2016) - 2-Layer Norm HyperLSTM	14.41M	1.219
Two layer LSTM	6.57M	1.243
Two Layer with New Cell	6.57M	1.228
Two Layer with New Cell	16.28M	1.214

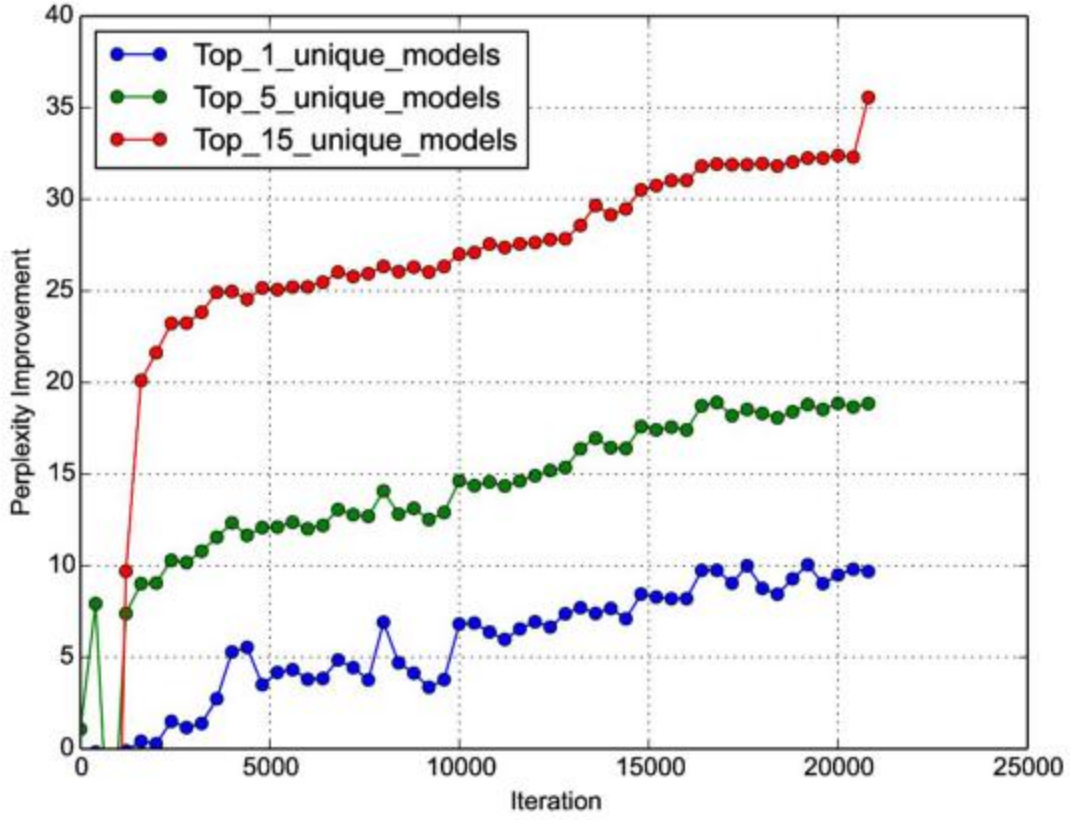
Experiments & Results

- **Control Experiment 1**
- **Adding more functions in the search space to test robustness**



Experiments & Results

- **Control Experiment 2**
- **Comparison against Random Search**
- Instead of policy gradient one can use Random Search to find the best network.



Conclusion

- Neural Architecture Search, an idea of using a recurrent neural network to compose neural network architectures.
- By using recurrent network as the controller, our method is flexible so that it can search variable-length architecture space.
- This method has strong empirical performance on very challenging benchmarks and presents a new research direction for automatically finding good neural network architectures.


thank you
danke
gracias
merci
obrigado
ngiyabonga
teşekkür ederim
謝謝
рахмат
Баярлалаа
спасибо
mersi
barka
welalin
tack
spas
kia ora
dank je
faafetai lava
vinaka
blagodaram
спасібі
misaotra
matondo
paldies
grazzi
mahalo
tapadh leat
хвала
asante
manana
obrigada
tenki
murakoze
shokrane
kiitos
dankie
dhanyavad
nanni
nandri
bedankt
bayarlalaa
gracie
hvala
maururu
köszönöm
enkosi
dziękuję
akun
dankon
aciū
djiere dieuf
tau
mochchakkeram
дякую
mamnun
chnorakaloutioun
grazias ago
gracies
sulpay
go raibh maith agat
sobodi
dėkuji
sağolun
sukriya
kop khun krap
taiku
arigatô
takk
dakujem
trugarez
mesi
didi madloba
najis tuke
ありがとう
tanemirt
grazie
rahmet
kam sah hamnida
rahmat
terima kasih
ευχαριστώ
diolch
dhanyavadagalu
shukriya
merce
мерси
감사합니다
xiexie