

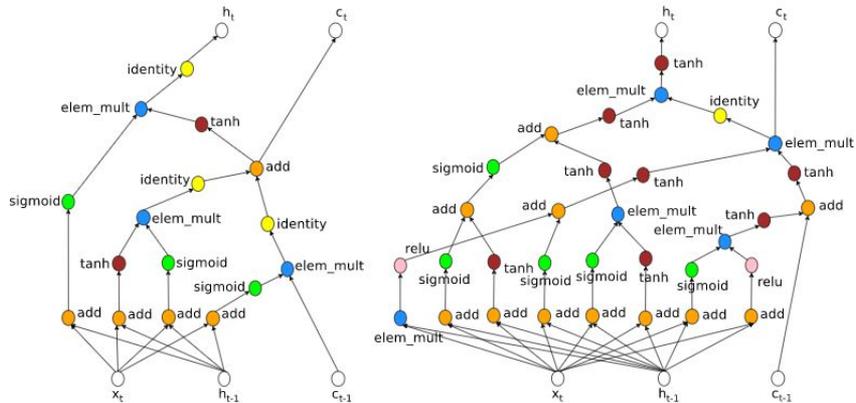


Neural Architecture Search with Reinforcement Learning

Quoc Le & Barret Zoph

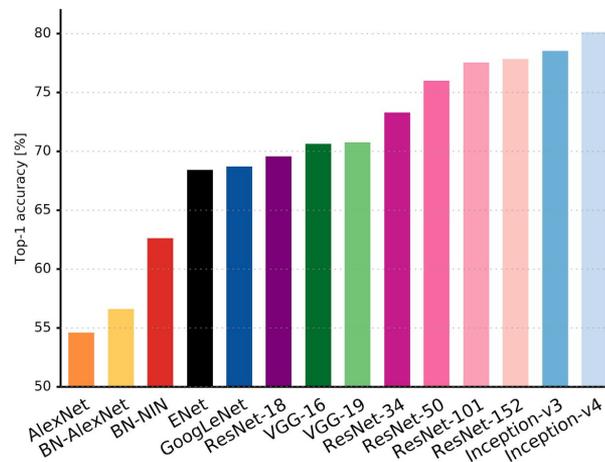
Thanks:

Vijay Vasudevan, Irwan Bello,
Jon Shlens, Google Brain team



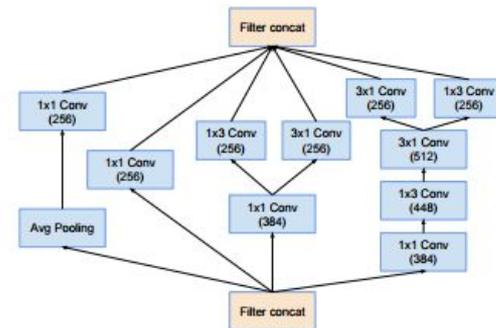
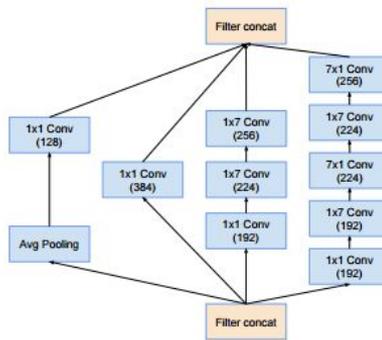
Motivation for Architecture Search

- Designing neural network architectures is hard
- Lots of human efforts go into tuning them
- There is not a lot of intuition into how to design them well
- Can we try and learn good architectures automatically?



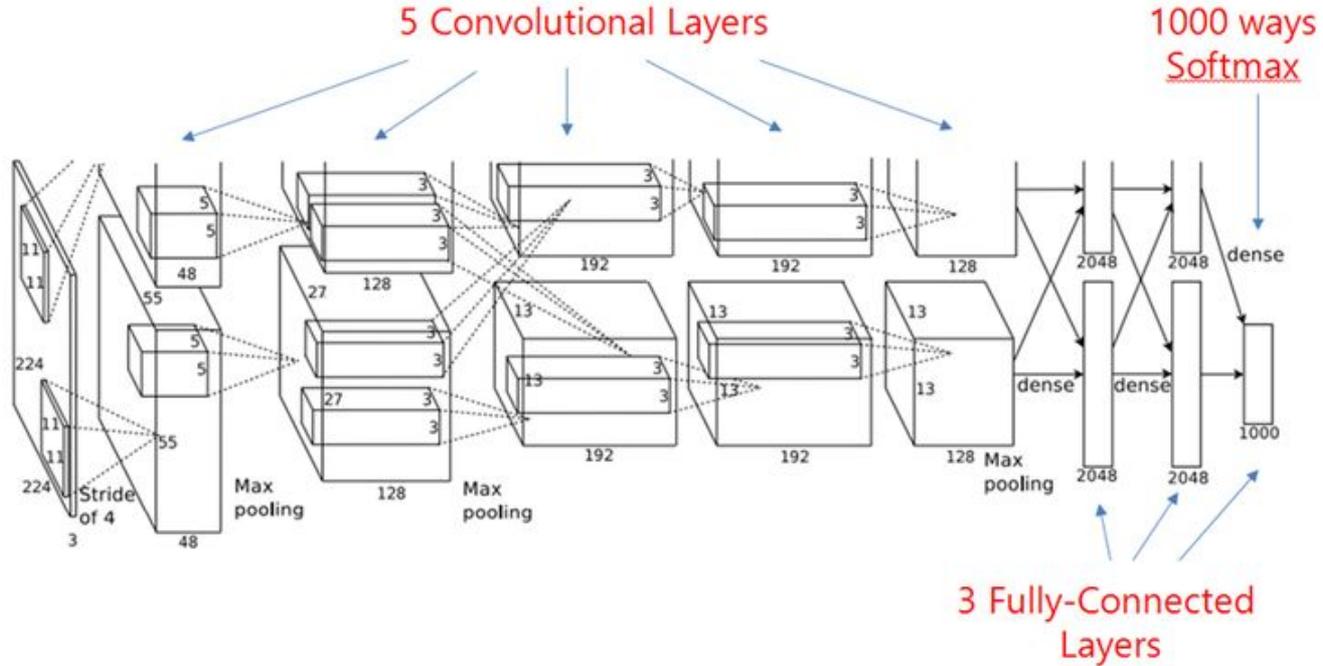
Google

Canziani et al, 2017



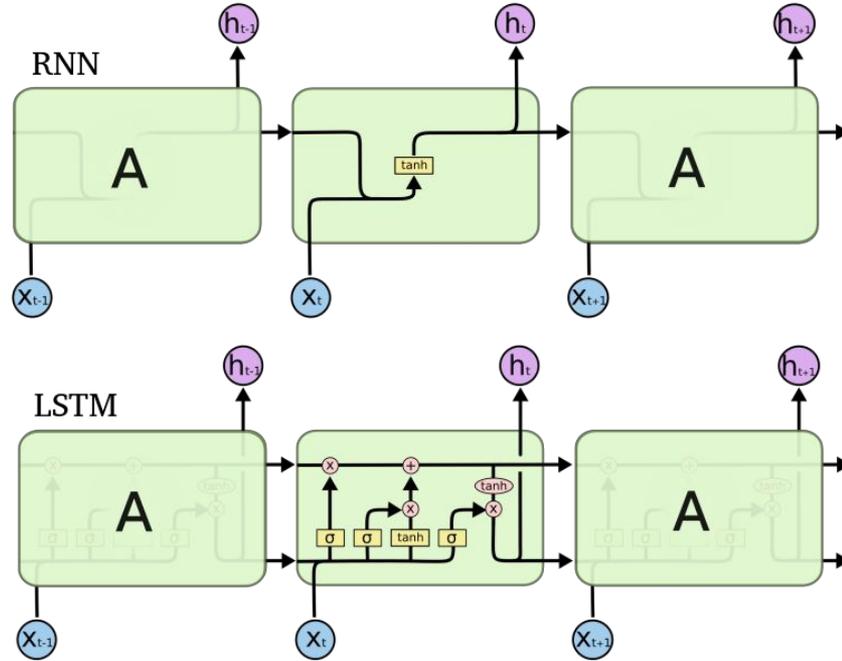
Two layers from the famous Inception V4 computer vision model.
Szegedy et al, 2017

Convolutional Architectures



Krizhevsky et al, 2012

Recurrent Architectures



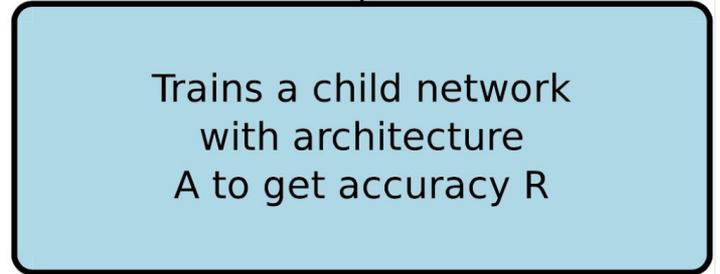
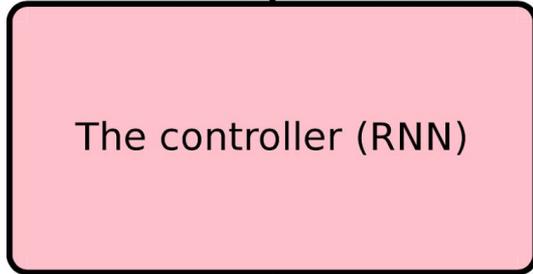
Hochreiter & Schmidhuber 1997

Neural Architecture Search

- Key idea is that we can specify the structure and connectivity of a neural network by using a configuration string
 - ["Filter Width: 5", "Filter Height: 3", "Num Filters: 24"]
- Our idea is to use a RNN ("Controller") to generate this string that specifies a neural network architecture
- Train this architecture ("Child Network") to see how well it performs on a validation set
- Use reinforcement learning to update the parameters of the Controller model based on the accuracy of the child model

Neural Architecture Search

Sample architecture A
with probability p

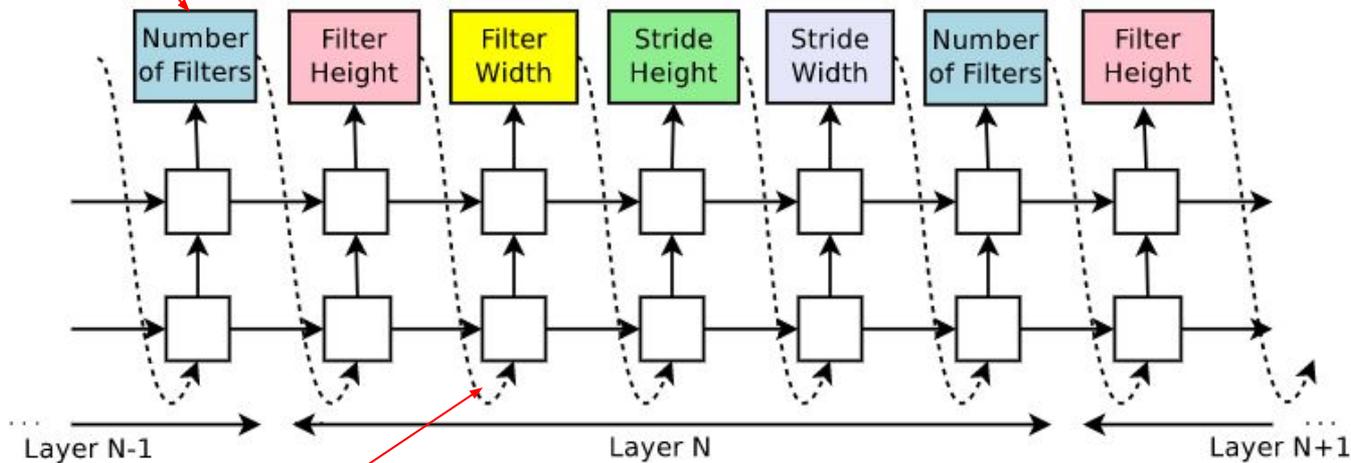


Compute gradient of p and
scale it by R to update
the controller

Neural Architecture Search for Convolutional Networks

Softmax classifier

Controller RNN



Embedding

Training with REINFORCE

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

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

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

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

Training with REINFORCE

Parameters of Controller RNN

Accuracy of architecture on held-out dataset

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

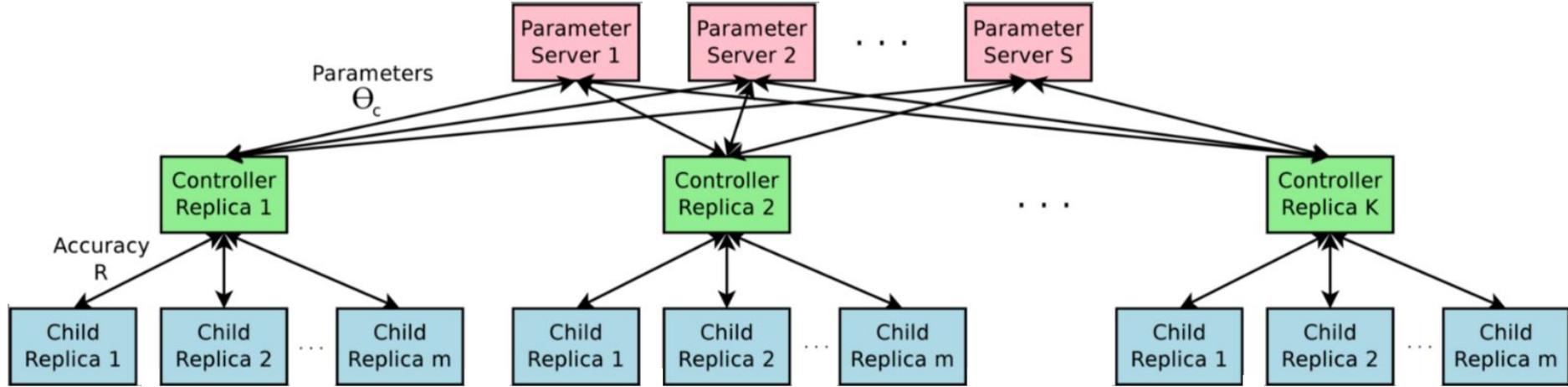
Architecture predicted by the controller RNN viewed as a sequence of actions

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

Number of models in minibatch

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

Distributed Training



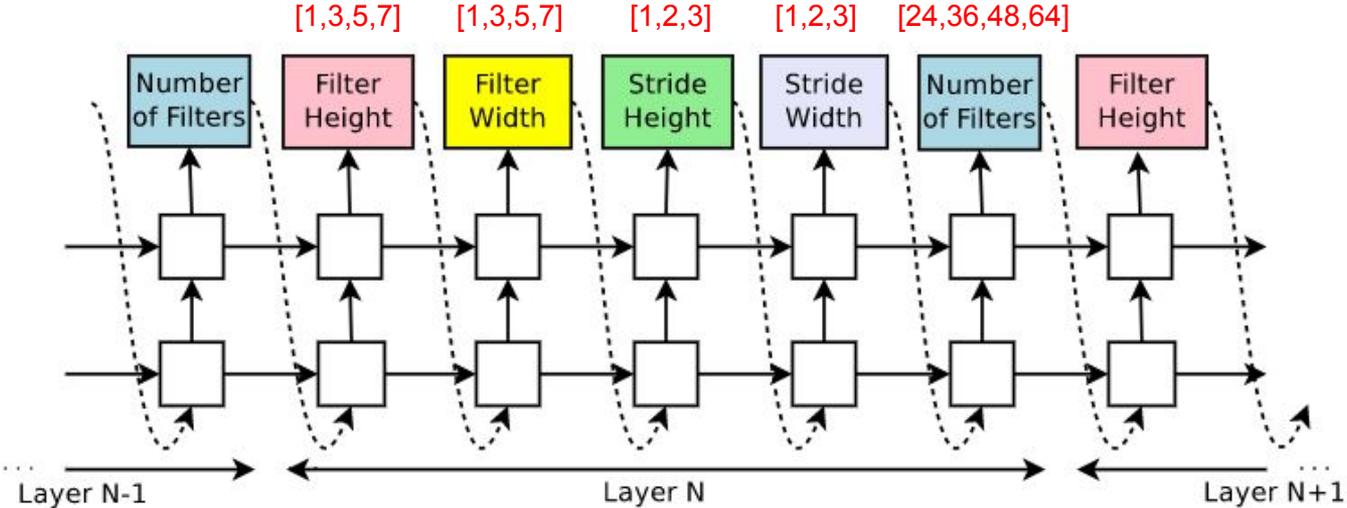
Overview of Experiments

- Apply this approach to Penn Treebank and CIFAR-10
- Evolve a convolutional neural network on CIFAR-10 and a recurrent neural network cell on Penn Treebank
- Achieve SOTA on the Penn Treebank dataset and almost SOTA on CIFAR-10 with a smaller and faster network
- Cell found on Penn Treebank beats LSTM baselines on other language modeling datasets and on machine translation

Neural Architecture Search for CIFAR-10

- We apply Neural Architecture Search to predicting convolutional networks on CIFAR-10
- Predict the following for a fixed number of layers (15, 20, 13):
 - Filter width/height
 - Stride width/height
 - Number of filters

Neural Architecture Search for CIFAR-10



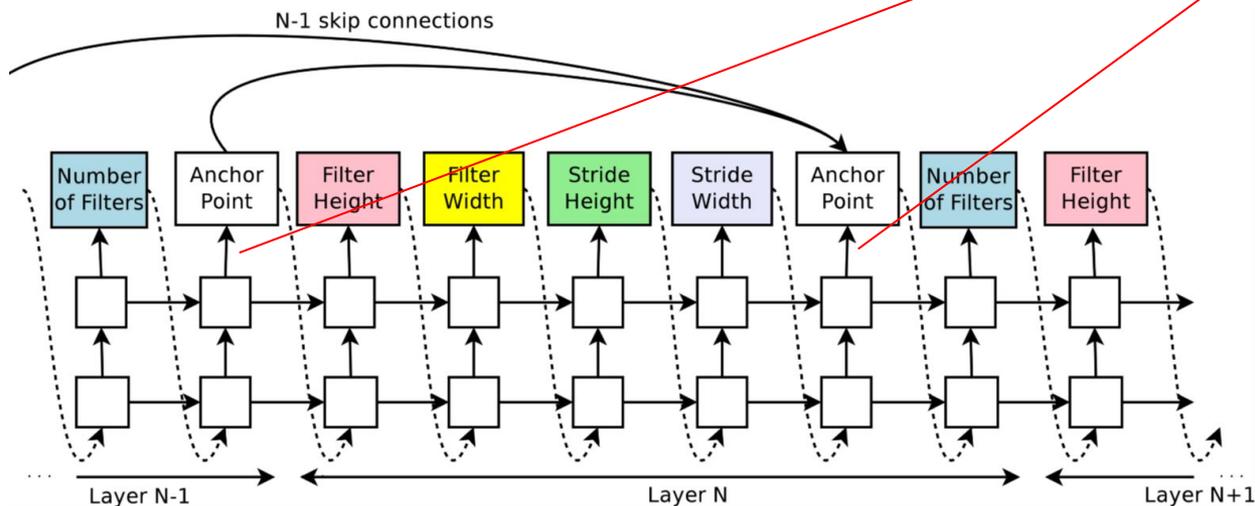
CIFAR-10 Prediction Method

- Expand search space to include branching and residual connections
- Propose the prediction of skip connections to expand the search space
- At layer N , we sample from $N-1$ sigmoids to determine what layers should be fed into layer N
- If no layers are sampled, then we feed in the minibatch of images
- At final layer take all layer outputs that have not been connected and concatenate them

Neural Architecture Search for CIFAR-10

$$P(\text{Layer } j \text{ is an input to layer } i) = \text{sigmoid}(v^T \tanh(W_{prev} * h_j + W_{curr} * h_i))$$

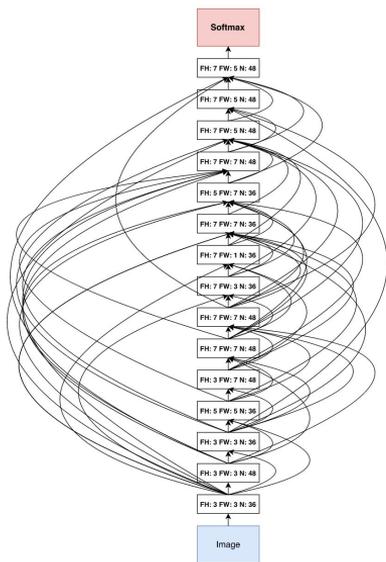
Weight Matrices



CIFAR-10 Experiment Details

- Use 100 Controller Replicas each training 8 child networks concurrently
- Method uses 800 GPUs concurrently at one time
- Reward given to the Controller is the maximum validation accuracy of the last 5 epochs squared
- Split the 50,000 Training examples to use 45,000 for training and 5,000 for validation
- Each child model was trained for 50 epochs
- Run for a total of **12,800** child models
- Used curriculum training for the Controller by gradually increasing the number of layers sampled

Neural Architecture Search for CIFAR-10



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

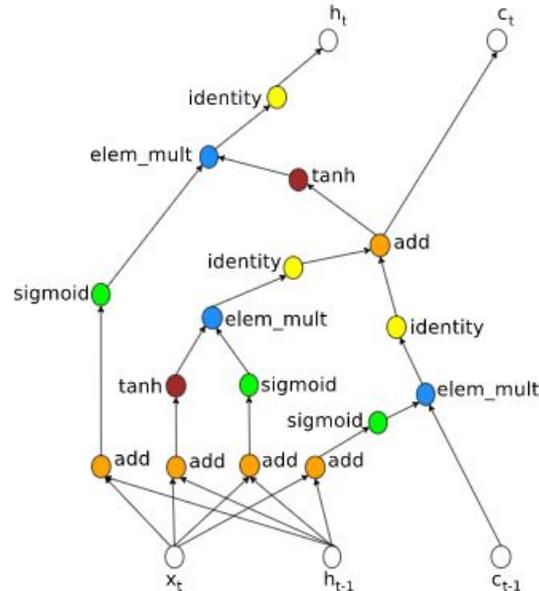
5% faster

Best result of evolution (Real et al, 2017): 5.4%

Best result of Q-learning (Baker et al, 2017): 6.92%

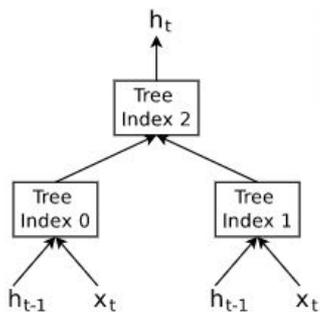
Recurrent Cell Prediction Method

- Created a search space for search over RNN cells like the LSTM or GRU
- Based our search space off the LSTM cell in that we have a recurrent state and cell

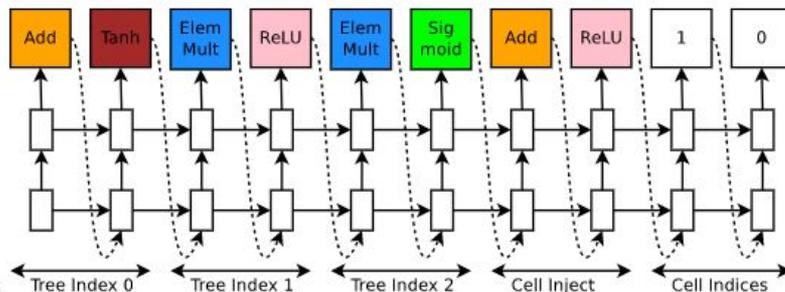


Recurrent Cell Prediction Method

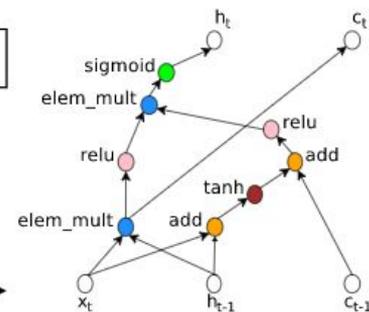
Cell Search Space



Controller RNN



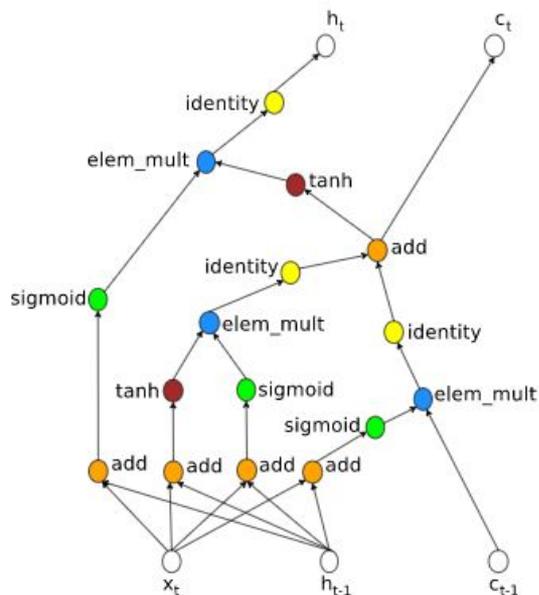
Created New Cell



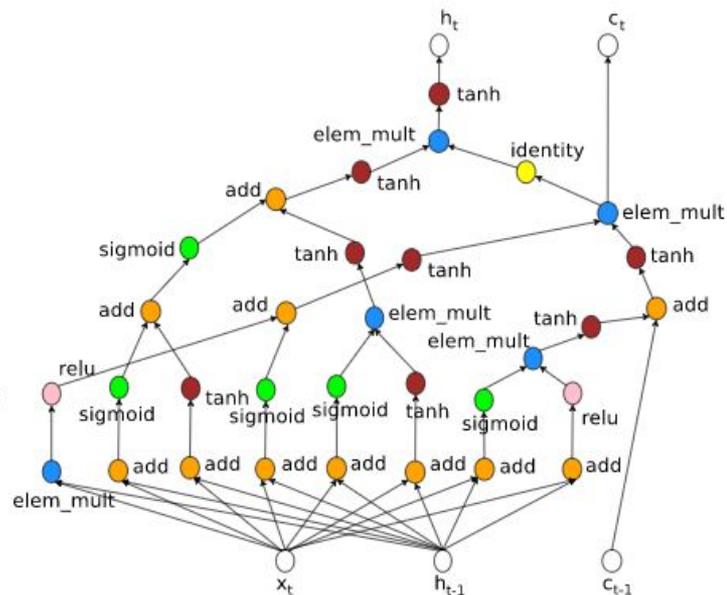
Penn Treebank Experiment Details

- Run Neural Architecture Search with our cell prediction method on the Penn Treebank language modeling dataset
- Previous Diagram had a base of 2, in this experiment we used a base of 8
- Use 400 Controller Replicas each training 1 child network
- Use 400 CPUs concurrently at one time
- Run for a total of **15,000** child models
- Reward for the Controller is $c/(\text{validation perplexity})^2$

Penn Treebank Results



LSTM Cell



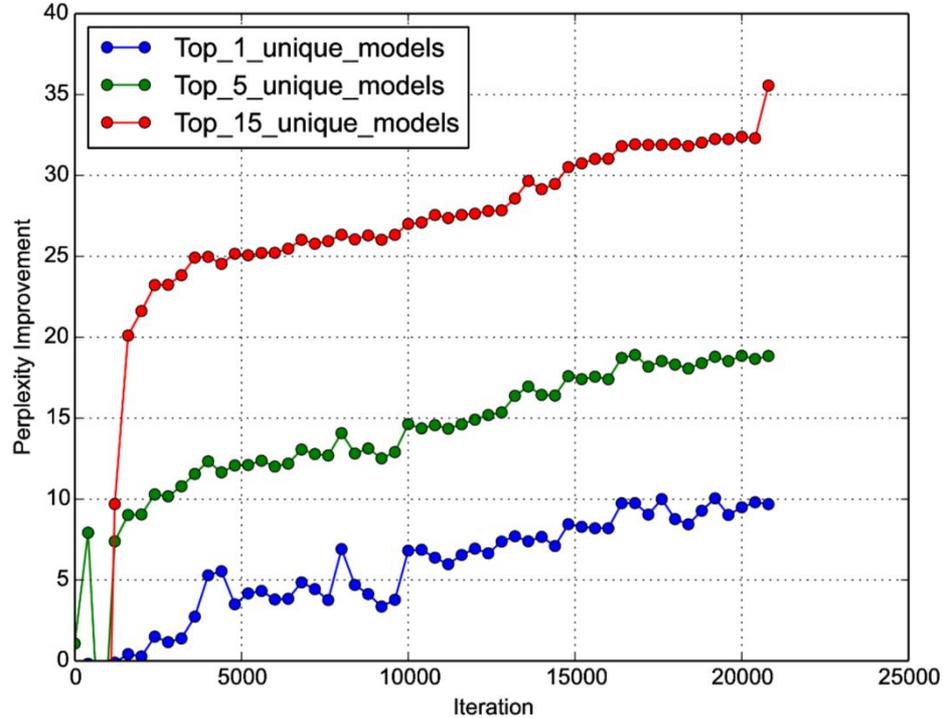
Neural Architecture Search (NAS) Cell

Penn Treebank 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

Comparison to Random Search

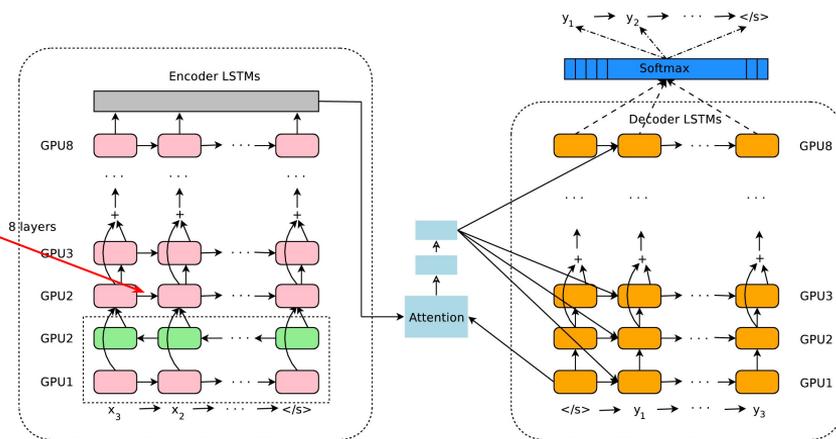


Transfer Learning on Character Level Language Modeling

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

Transfer Learning on Neural Machine Translation

LSTM Cell



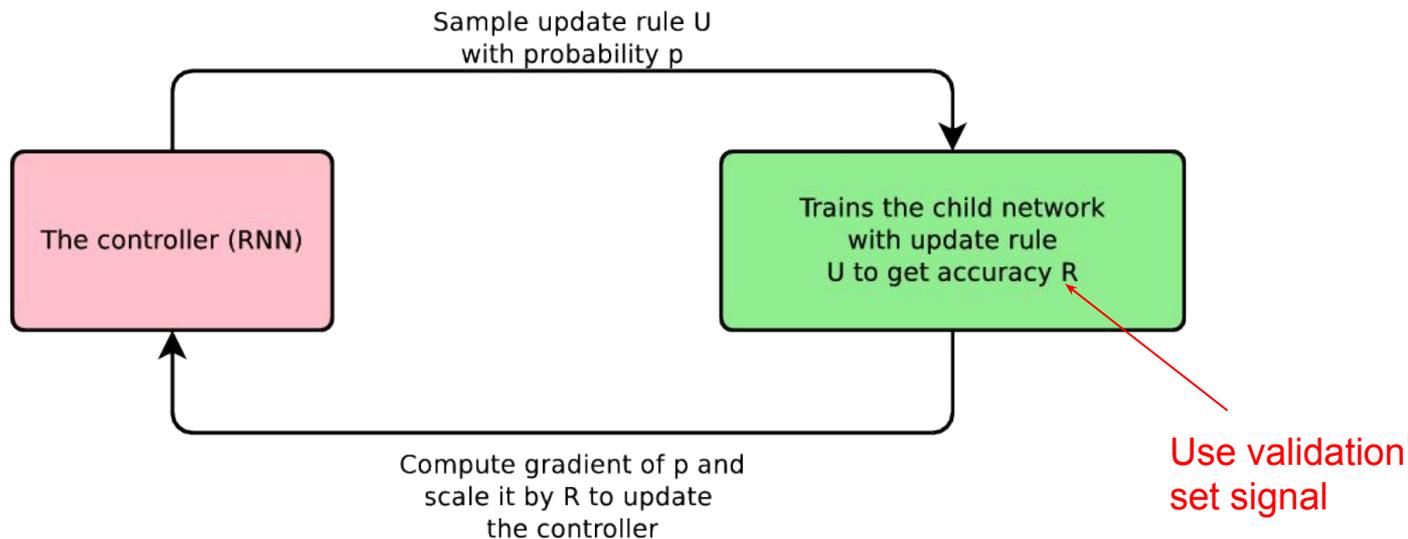
Google Neural Machine Translation
(Wu et al, 2016)

Model	WMT'14 en->de Test Set BLEU
GNMT LSTM	24.1
GNMT NAS Cell	24.6

Neural Optimizer Search

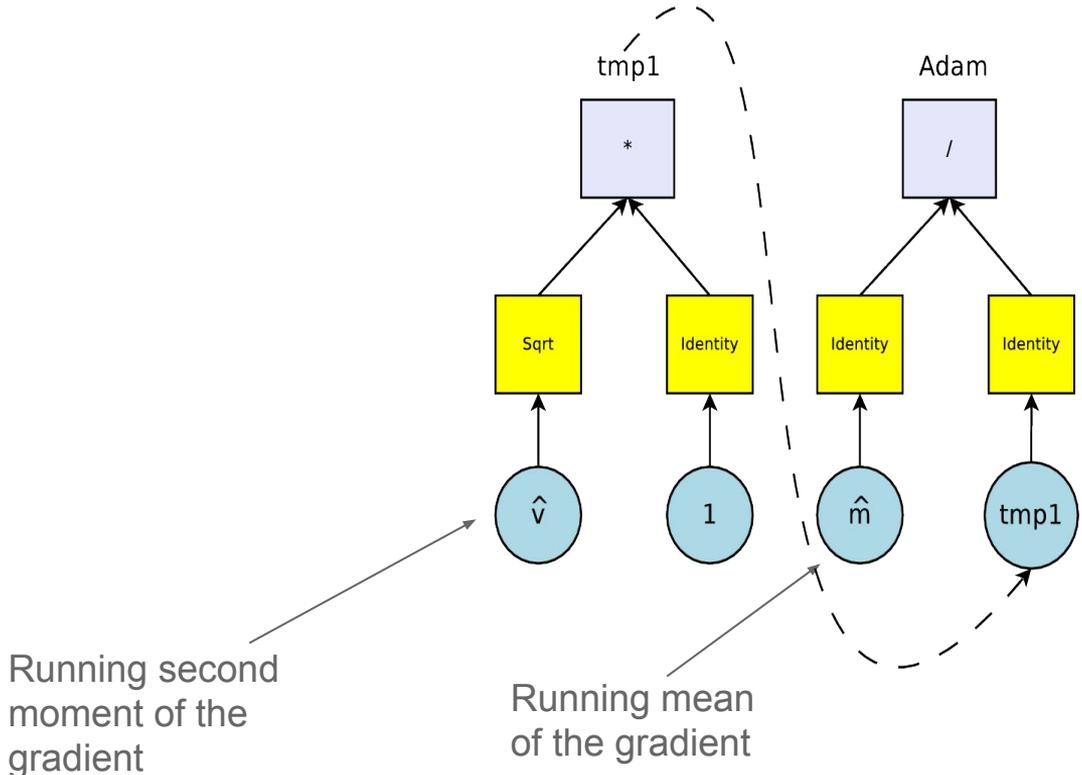
- Optimizers are also hard to design just like neural architectures
- Many different optimizers exist such as ADAM, RMSProp, ADADelta, Momentum, SGD, etc..
- We can use the previous method to also search over optimizers
- Search over optimizers given a fixed neural network architecture and dataset

Neural Optimizer Search

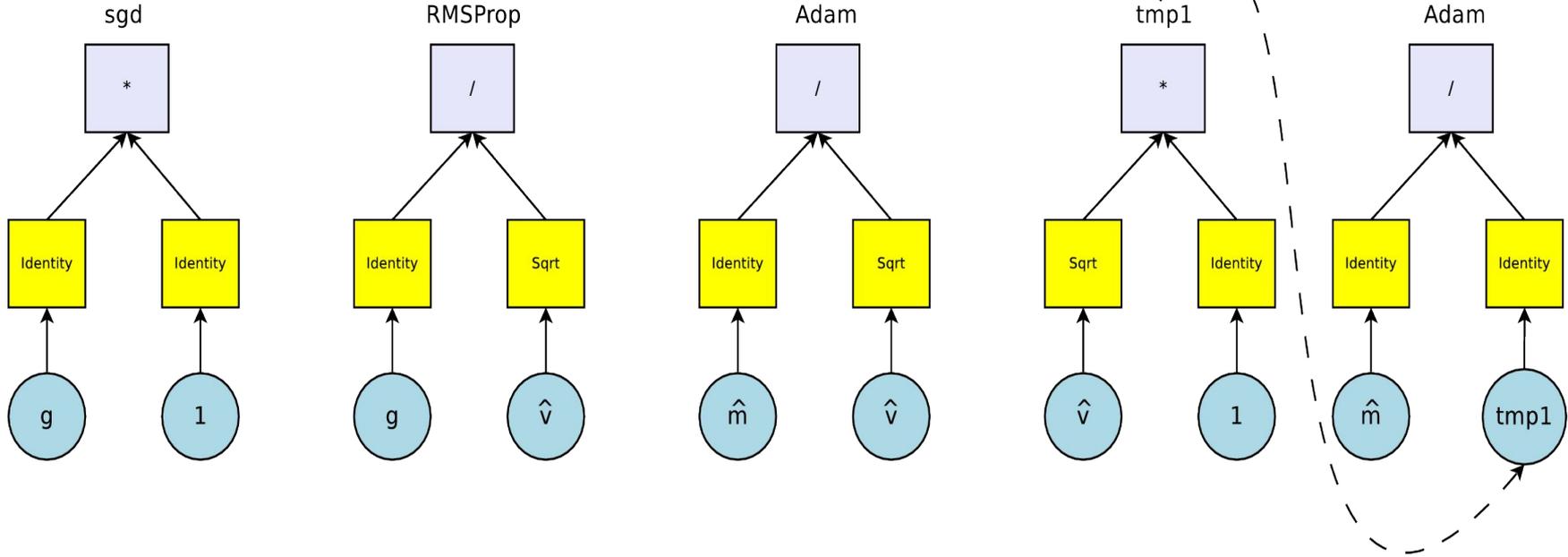


Adam Optimizer

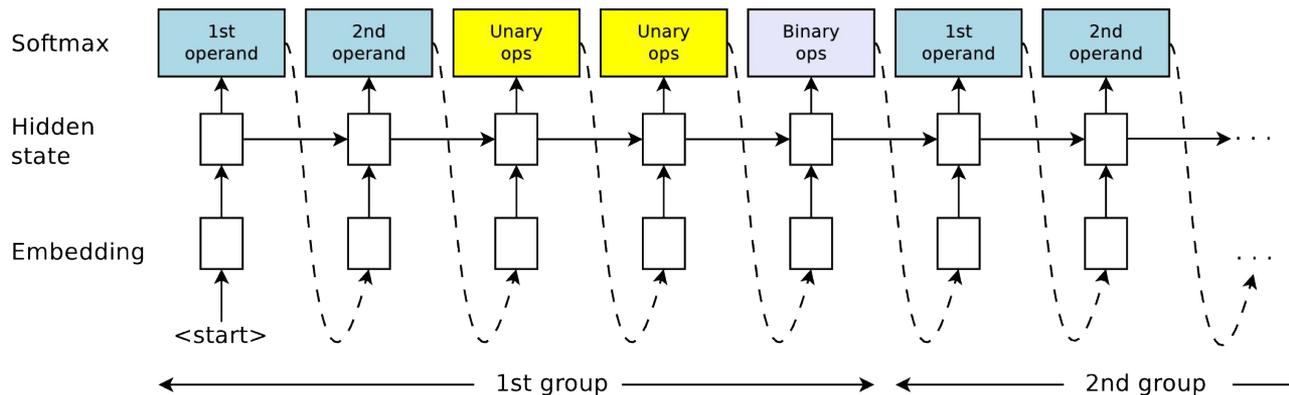
Computation graph of the Adam optimizer (Kingma & Ba, 2015)



Commonly Used Neural Optimizers



Neural Search for Optimizers



The controller iteratively selects subsequences of length 5.

- First selects the 1st and 2nd operands $op1$ and $op2$
- Selects 2 unary functions $u1$ and $u2$ to apply to the operands
- Selects a binary function b that combines the outputs of the unary functions.
- The resulting $b(u1(op1), u2(op2))$ then becomes an operand that can be selected in the subsequent group of predictions or becomes the update rule.

Designing Search Spaces - Operands & Functions

- g, g^2, g^3
- (bias-corrected) moving averages
- $\text{sign}(g), \text{sign}(\text{moving average})$
- Constant
- Constant noise
- Annealed noise
- Weight
- ADAM, RMSProp
- Cyclical learning rates
- Restart learning rates
- ...

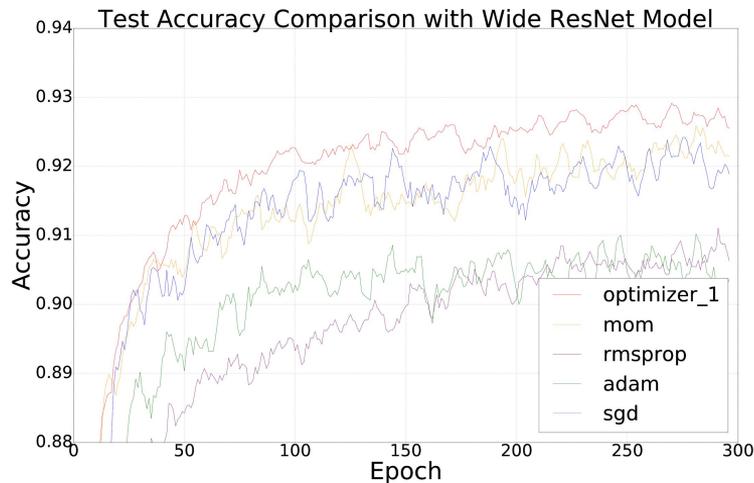
- Exp
- Log
- Sqrt
- $\text{clip}(\cdot, l)$
- $\text{drop}(\cdot, p)$
- ...

- Addition
- Subtraction
- Multiplication
- Division
- Keep left
- Exponentiation
- Max
- Min

CIFAR-10 Wide ResNet

Optimizer	Final Val	Final Test	Best Val	Best Test
SGD	92.0	91.8	92.9	91.9
Momentum	92.7	92.1	93.1	92.3
ADAM	90.4	90.1	91.8	90.7
RMSProp	90.7	90.3	91.4	90.3
$[e^{\text{sign}(g) * \text{sign}(m)} + \text{clip}(g, 10^{-4})] * g$	92.5	92.4	93.8	93.1
$\text{clip}(\hat{m}, 10^{-4}) * e^{\hat{v}}$	93.5	92.5	93.8	92.7
$\hat{m} * e^{\hat{v}}$	93.1	92.4	93.8	92.6
$g * e^{\text{sign}(g) * \text{sign}(m)}$	93.1	92.8	93.8	92.8
$\text{drop}(g, 0.3) * e^{\text{sign}(g) * \text{sign}(m)}$	92.7	92.2	93.6	92.7
$\hat{m} * e^{g^2}$	93.1	92.5	93.6	92.4
$\text{drop}(\hat{m}, 0.1) / (e^{g^2} + \epsilon)$	92.6	92.4	93.5	93.0
$\text{drop}(g, 0.1) * e^{\text{sign}(g) * \text{sign}(m)}$	92.8	92.4	93.5	92.2
$\text{clip}(\text{RMSProp}, 10^{-5}) + \text{drop}(\hat{m}, 0.3)$	90.8	90.8	91.4	90.9
$\text{ADAM} * e^{\text{sign}(g) * \text{sign}(m)}$	92.6	92.0	93.4	92.0
$\text{ADAM} * e^{\hat{m}}$	92.9	92.8	93.3	92.7
$g + \text{drop}(\hat{m}, 0.3)$	93.4	92.9	93.7	92.9
$\text{drop}(\hat{m}, 0.1) * e^{g^3}$	92.8	92.7	93.7	92.8
$g - \text{clip}(g^2, 10^{-4})$	93.4	92.8	93.7	92.8
$e^g - e^{\hat{m}}$	93.2	92.5	93.5	93.1
$\text{drop}(\hat{m}, 0.3) * e^w$	93.2	93.0	93.5	93.2

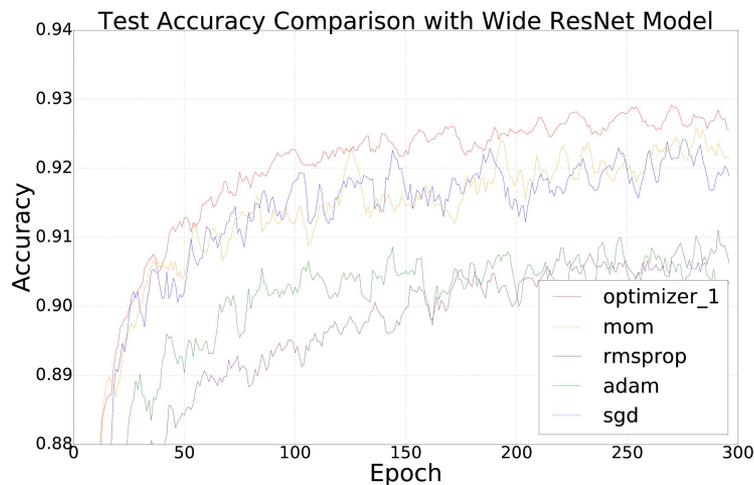
Table 1. Performance of Neural Optimizer Search and standard optimizers on the Wide-ResNet architecture (Zagoruyko & Komodakis, 2016) on CIFAR-10. Final Val and Final Test refer to the final validation and test accuracy after for training for 300 epochs. Best Val corresponds to the best validation accuracy over the 300 epochs and Best Test is the test accuracy at the epoch where the validation accuracy was the highest.



CIFAR-10 Wide ResNet

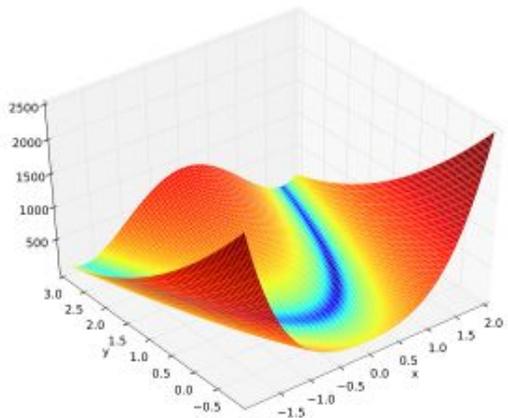
Optimizer	Final Val	Final Test	Best Val	Best Test
SGD	92.0	91.8	92.9	91.9
Momentum	92.7	92.1	93.1	92.3
ADAM	90.4	90.1	91.8	90.7
RMSProp	90.7	90.3	91.4	90.3
$[e^{\text{sign}(g) * \text{sign}(m)} + \text{clip}(g, 10^{-4})] * g$	92.5	92.4	93.8	93.1
$\text{clip}(\hat{m}, 10^{-4}) * e^{\hat{v}}$	93.5	92.5	93.8	92.7
$\hat{m} * e^{\hat{v}}$	93.1	92.4	93.8	92.6
$g * e^{\text{sign}(g) * \text{sign}(m)}$	93.1	92.8	93.8	92.8
$\text{drop}(g, 0.3) * e^{\text{sign}(g) * \text{sign}(m)}$	92.7	92.2	93.6	92.7
$\hat{m} * e^{g^2}$	93.1	92.5	93.6	92.4
$\text{drop}(\hat{m}, 0.1) / (e^{g^2} + \epsilon)$	92.6	92.4	93.5	93.0
$\text{drop}(g, 0.1) * e^{\text{sign}(g) * \text{sign}(m)}$	92.8	92.4	93.5	92.2
$\text{clip}(\text{RMSProp}, 10^{-5}) + \text{drop}(\hat{m}, 0.3)$	90.8	90.8	91.4	90.9
$\text{ADAM} * e^{\text{sign}(g) * \text{sign}(m)}$	92.6	92.0	93.4	92.0
$\text{ADAM} * e^{\hat{m}}$	92.9	92.8	93.3	92.7
$g + \text{drop}(\hat{m}, 0.3)$	93.4	92.9	93.7	92.9
$\text{drop}(\hat{m}, 0.1) * e^{g^3}$	92.8	92.7	93.7	92.8
$g - \text{clip}(g^2, 10^{-4})$	93.4	92.8	93.7	92.8
$e^g - e^{\hat{m}}$	93.2	92.5	93.5	93.1
$\text{drop}(\hat{m}, 0.3) * e^w$	93.2	93.0	93.5	93.2

Table 1. Performance of Neural Optimizer Search and standard optimizers on the Wide-ResNet architecture (Zagoruyko & Komodakis, 2016) on CIFAR-10. Final Val and Final Test refer to the final validation and test accuracy after for training for 300 epochs. Best Val corresponds to the best validation accuracy over the 300 epochs and Best Test is the test accuracy at the epoch where the validation accuracy was the highest.

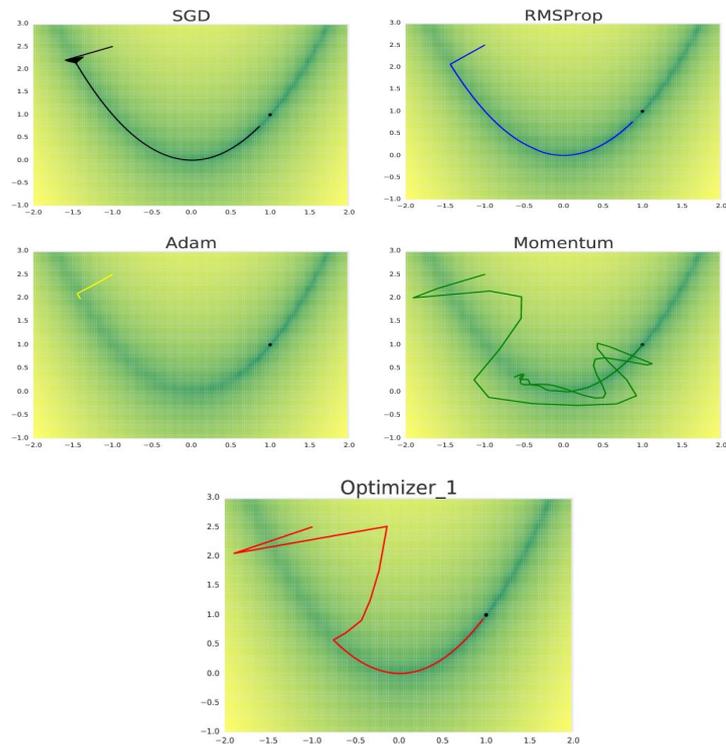


Control experiment with Rosenbrock function

Common stochastic optimization unit test



- Valley is easy to find
- Global minimum is hard to find



Neural Machine Translation

- Google Neural Machine Translation (Wu et al, 2016)
- All hyperparameters were tuned for the Adam optimizer (Wu et al, 2016)
- 0.5 BLEU improvement on WMT English to German task

Optimizer	Train Perplexity	Test BLEU
Adam	1.49	24.5
$g * e^{\text{sign}(g) * \text{sign}(m)}$	1.39	25.0