Meta Learning

Jakub Střelský

Machine learning

Parametric machine learning algorithms:

1. Define parametric model
2. Learn the model parameters from training data

Step 2 is typically a form of function optimization

(e.g. maximizing conditional likelihood of parameters given the training data)

How do we make it work

- Design models that describes data well and can be learned efficiently - very important
  - We cannot recover from poor choice of model

model complexity →

Cannot describe data	“Just right”	Overfitting and infeasible to train

  - Choice of model must reflect complexity of data
- Apply proper learning algorithm to find parameters of selected model
- Fine-tune learning algorithm (e.g. find good hyper-parameters for given learning instance)

Meta learning

- Simply: Learning to learn
- Training data are instances of “similar” learning problems
- We want to make use of learning experience in order to improve learning in future

How ?

- Typical example: tuning of hyper-parameters of learning
- But even: altering learning algorithm or model

When to consider meta learning

- If we assume that learning instances are related, but the relation is subtle and hard to describe mathematically

Linear regression

Image classification with neural networks

Neural optimizer search with reinforcement learning

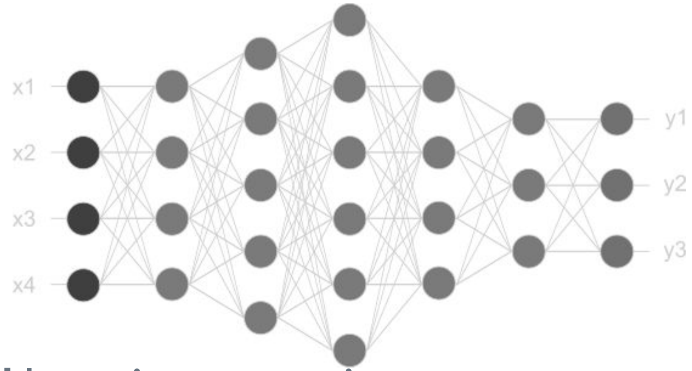
Irwan Bello, Barret Zoph, Vijay Vasudevan and Quoc V. Le

Published 2017 in ICML

http://proceedings.mlr.press/v70/bello17a.html

Neural networks

- Neural network represents function
  f: $\mathbf{X} \times \mathbf{\Theta} \rightarrow \mathbf{Y}$

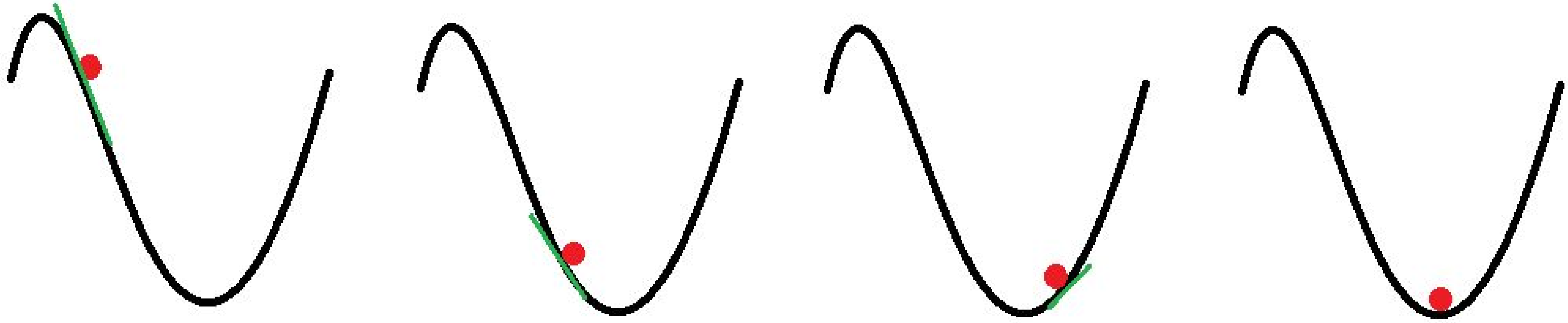


- In **supervised** learning scenario,
  we have a set of input-target pairs (x_i, y_i) $i = 1,2...N$
- Objective function J defined for a task, e.g. MSE for regression:

$$J = (1/N) \sum_i (f(\mathbf{x}_i, \boldsymbol{\theta}) - \mathbf{y}_i)^2$$

Neural networks training

- Network is trained by searching minimum of J
- We calculate gradient $\nabla_{\theta} J$ (backpropagation)
- $\theta_{new} = \theta - \lambda * \nabla_{\theta} J$



Tricks

- Mini-batches
- Decaying learning rate
- Stabilizing updates
- E.g Adam (roughly):

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \lambda_t * \mathbf{m}_t / \mathrm{sqrt}(\mathbf{v}_t)$$

$\mathbf{m}_t$: estimate of gradient mean
$\mathbf{v}_t$: estimate of gradient variance

Learning optimizers

One step of (meta) learning cycle:

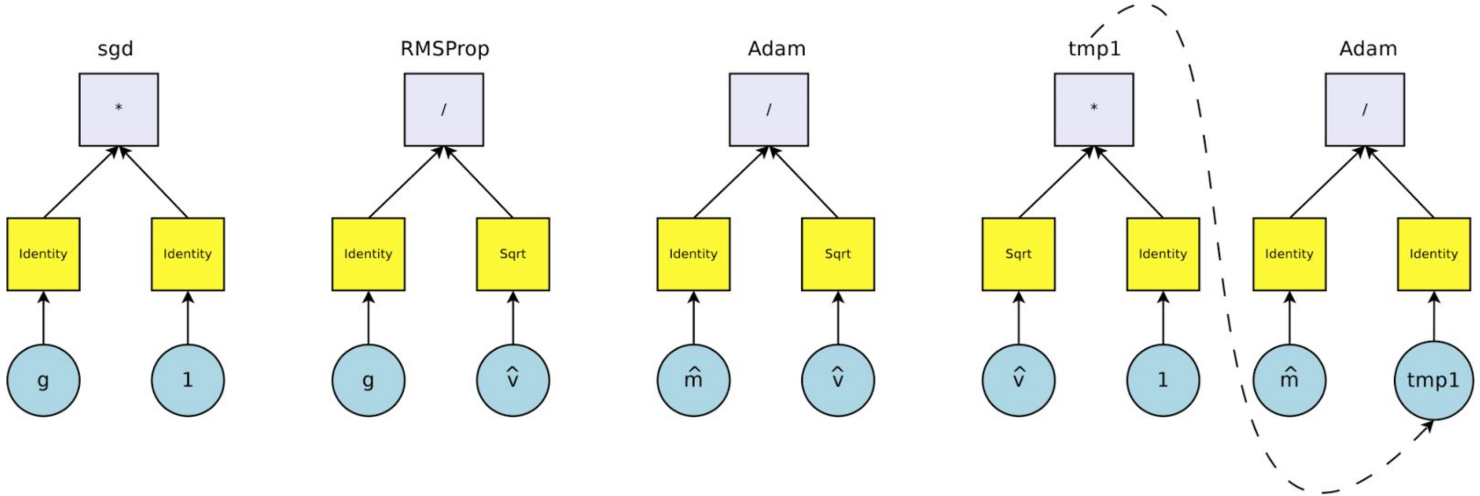
- Controller generates update rule $\Delta\theta$ of optimizer
- We train neural network using $\Delta\theta$ $(\theta_{t+1} = \theta_t - \Delta\theta_t)$
- Reward of $\Delta\theta$ is expected accuracy of neural network on validation data

Rules

- Rules are expressions defined by binary tree
- $\Delta\theta = \lambda * b(u_1(op_1), u_2(op_2))$

  b-binary op, $u_{1,2}$ - unary ops, $op_{1,2}$ - operands

  Operands are either inputs or expressions



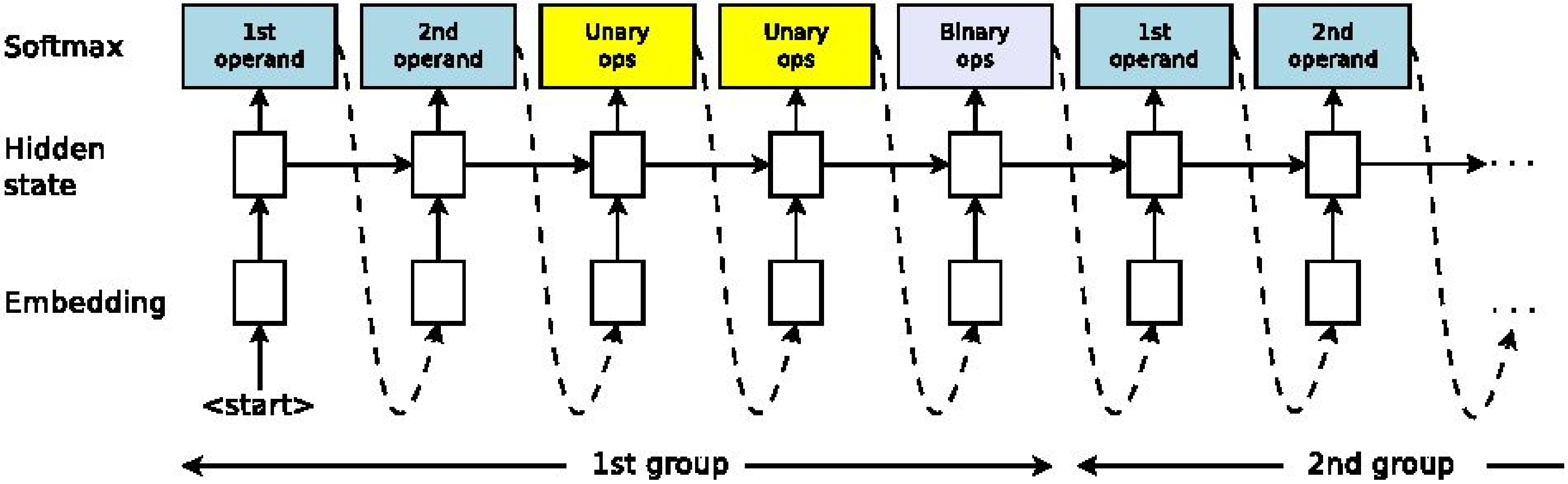
Rules

- **Operands:**
  gradient, estimated moments of gradient,
  sign(gradient), Adam, RMSProp, small noise, constant…
- **Unary operations u(x):**
  x, -x, log(abs(x)), exp(x), sign(x), clip(x,0.001)...
- **Binary operations b(x,y):**
  Addition, subtraction, multiplication, division and b(x,y) = x

- Depth of trees was bounded by depths: 1,2 and 3

Controller

–



Softmax
Hidden state
Embedding
1st operand
2nd operand
Unary ops
Unary ops
Binary ops
1st operand
2nd operand
<start>
1st group
2nd group

Learning details

- Controller is learned via reinforcement learning (variant of policy gradient method)
- Target network is small convolutional network with 2 layers
- Target network is trained for 5 epochs on image classification dataset CIFAR-10
- Learning rate of update rule is determined by choosing best learning rate from 10^{-5}, 10^{-4}, … 10^{1} after 1 epoch

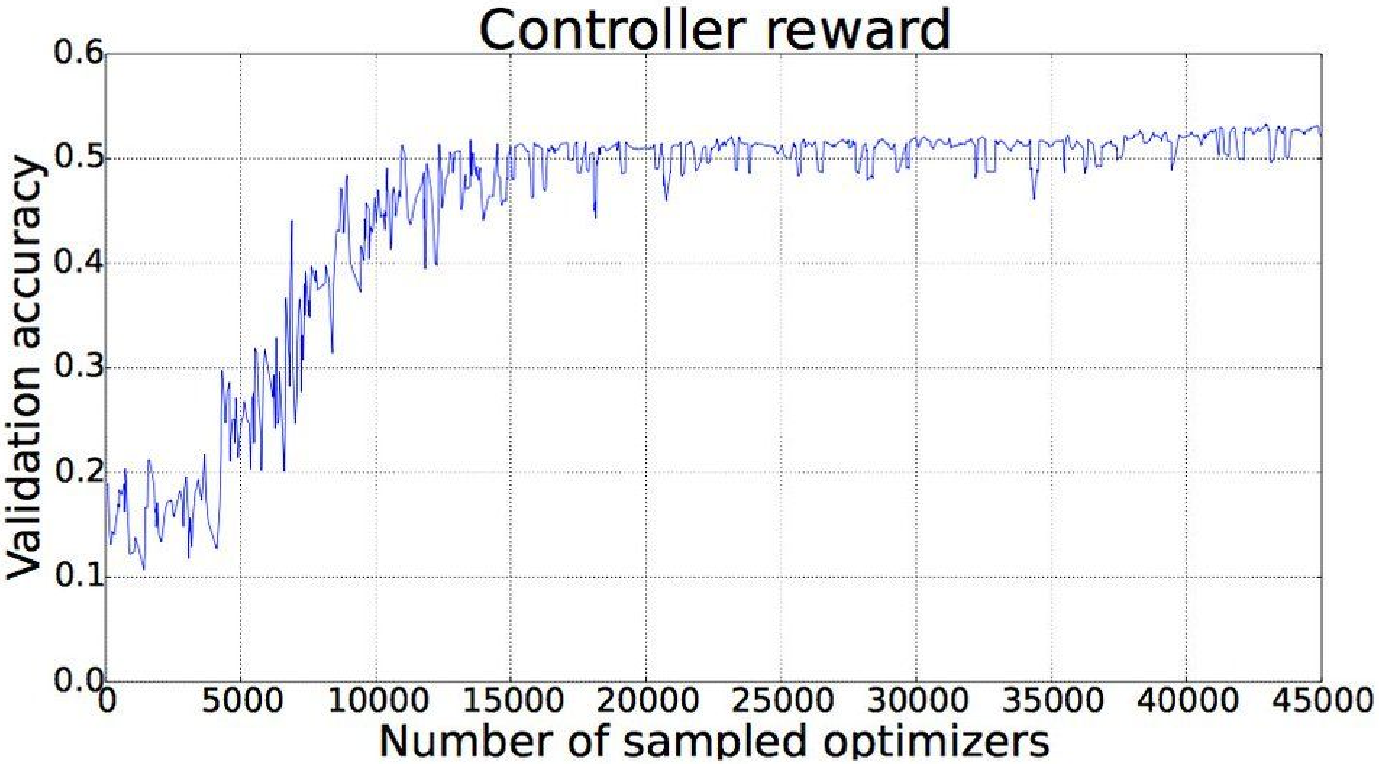


Figure 4. Controller reward increasing over time as more optimizers are sampled.

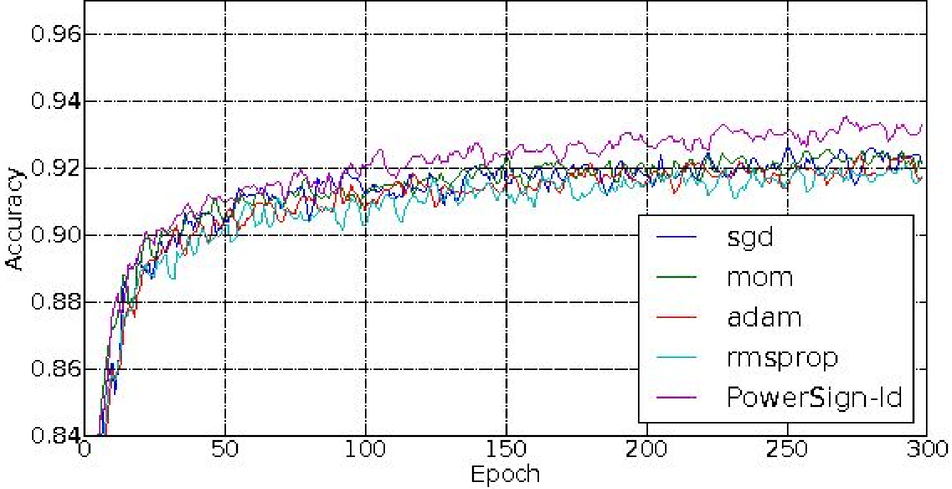
Discovered rules

Successful building block:

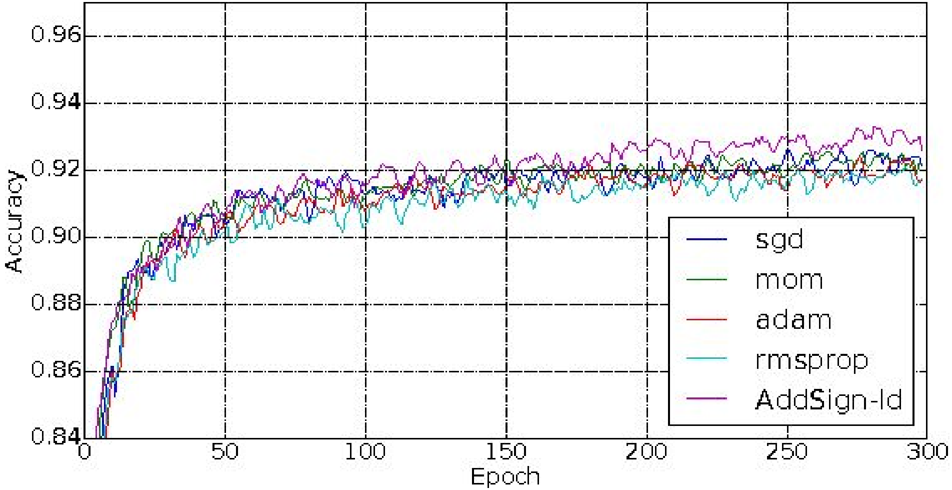
$$\mathbf{g} * \exp(\mathrm{sign}(\mathbf{g}) * \mathrm{sign}(\mathbf{m}))$$

Exp is positive, so weight updates follow direction -**g** with scaling. Scaling is either e when signs agree, or $1/e$ when signs disagree.

- $\mathbf{g} * (\mathrm{clip}(\mathbf{g}, 10^{-4}) + \exp(\mathrm{sign}(\mathbf{g}) * \mathrm{sign}(\mathbf{m}))$
- $\mathbf{Adam} * \exp(\mathrm{sign}(\mathbf{g}) * \mathrm{sign}(\mathbf{m}))$
- $\mathrm{drop}(\mathbf{g}, 0.1) * \exp(\mathrm{sign}(\mathbf{g}) * \mathrm{sign}(\mathbf{m}))$

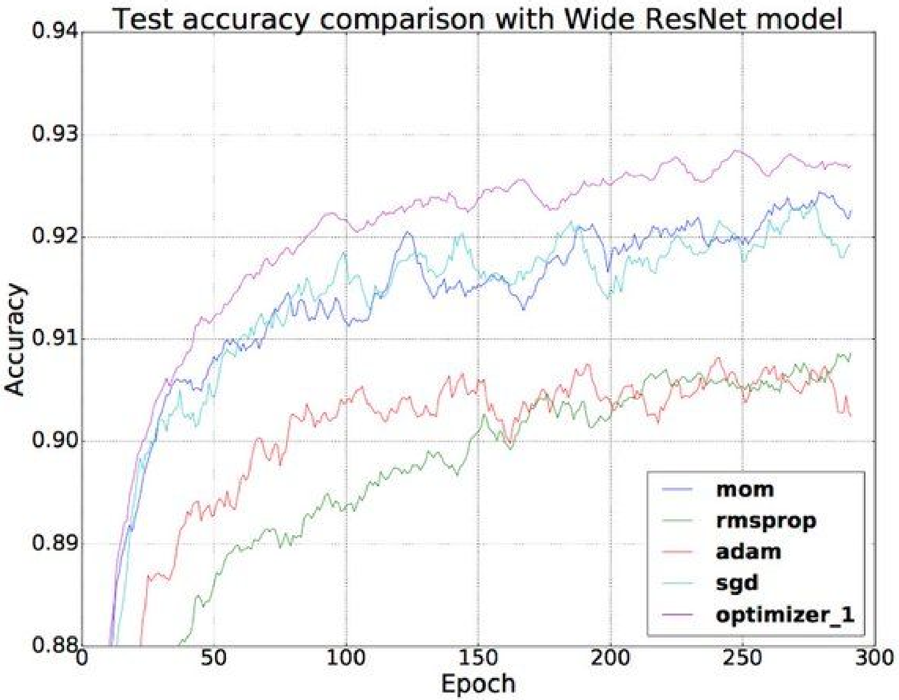


Accuracy
Epoch
0.84
0.86
0.88
0.90
0.92
0.94
0.96
0
50
100
150
200
250
300
sgd
mom
adam
rmsprop
PowerSign-ld



Accuracy
Epoch
0.84
0.86
0.88
0.90
0.92
0.94
0.96
0
50
100
150
200
250
300
sgd
mom
adam
rmsprop
AddSign-ld

CIFAR-10 with Wide ResNet



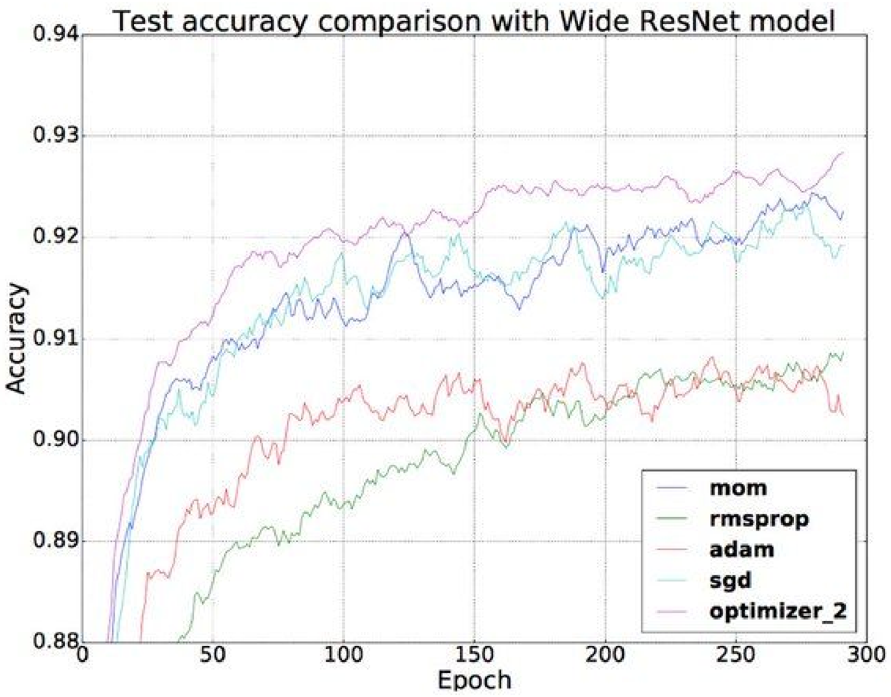


Figure 7. Comparison of two of the best optimizers found with Neural Optimizer Search using Wide ResNet as the architecture. Optimizer_1 refers to $[e^{\text{sign}(g)*\text{sign}(m)} + \text{clip}(g, 10^{-4})] * g$ and Optimizer_2 refers to $\text{drop}(\hat{m}, 0.3) * e^{10^{-3}w}$.

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
$\text{drop}(\hat{m}, 0.3) * e^{10^{-3}w}$	93.2	93.0	93.5	93.2

Table 1. Performance of Neural Search 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.

Final notes

- Rule $\mathbf{g} * \exp(\text{sign}(\mathbf{g}) * \text{sign}(\mathbf{m}))$ was also applied to language translation with RNN yielding better accuracy than Adam
- The rule is also more memory efficient than Adam (it does not need to store variance estimate)
- Overall very good application of meta learning (maybe yielding new “default” optimizer)

Thank you for attention!