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

<table>
<thead>
<tr>
<th>Cannot describe data</th>
<th>&quot;Just right&quot;</th>
<th>Overfitting and infeasible to train</th>
</tr>
</thead>
</table>
- Choice of model must reflect complexity of data
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 \( f: \mathbf{X} \times \Theta \rightarrow \mathbf{Y} \)

- In *supervised* learning scenario, we have a set of input-target pairs \((x_i, y_i)\) where \(i = 1, 2, \ldots, N\)

- Objective function \( J \) defined for a task, e.g. MSE for regression:

\[
J = \frac{1}{N} \sum_i (f(x_i, \Theta) - y_i)^2
\]
Neural networks training

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

- Mini-batches
- Decaying learning rate
- Stabilizing updates
- E.g Adam (roughly):
  \[
  \theta_{t+1} = \theta_t - \lambda_t \cdot \frac{m_t}{\sqrt{v_t}}
  \]
  
  $m_t$: estimate of gradient mean
  $v_t$: 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
- \( \Delta \theta = \lambda \ast b(u_1(op_1), u_2(op_2)) \)
  
  - \(b\) - 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** $u(x)$:
  $x$, $-x$, $\log(|x|)$, $e^x$, $\text{sign}(x)$, $\text{clip}(x, 0.001)$...

- **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
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 $10^{-5}, 10^{-4}, \ldots, 10^1$ after 1 epoch
Figure 4. Controller reward increasing over time as more optimizers are sampled.
Discovered rules

Successful building block:

\[ g \ast \exp(\text{sign}(g) \ast \text{sign}(m)) \]

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

- \( g \ast(\text{clip}(g,10^{-4}) + \exp(\text{sign}(g) \ast \text{sign}(m))) \)
- \( \text{Adam} \ast \exp(\text{sign}(g) \ast \text{sign}(m)) \)
- \( \text{drop}(g,0.1) \ast \exp(\text{sign}(g) \ast \text{sign}(m)) \)
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 \[e^{\text{sign}(g)\cdot\text{sign}(m)} + \text{clip}(g, 10^{-4})] \cdot g \text{ and Optimizer_2 refers to } \text{drop}(\hat{m}, 0.3) \cdot e^{10^{-3}\cdot w}.\]
<table>
<thead>
<tr>
<th>Optimizer</th>
<th>Final Val</th>
<th>Final Test</th>
<th>Best Val</th>
<th>Best Test</th>
</tr>
</thead>
<tbody>
<tr>
<td>SGD</td>
<td>92.0</td>
<td>91.8</td>
<td>92.9</td>
<td>91.9</td>
</tr>
<tr>
<td>Momentum</td>
<td>92.7</td>
<td>92.1</td>
<td>93.1</td>
<td>92.3</td>
</tr>
<tr>
<td>ADAM</td>
<td>90.4</td>
<td>90.1</td>
<td>91.8</td>
<td>90.7</td>
</tr>
<tr>
<td>RMSProp</td>
<td>90.7</td>
<td>90.3</td>
<td>91.4</td>
<td>90.3</td>
</tr>
<tr>
<td>$[e^{\text{sign}(g)\cdot\text{sign}(m)} + \text{clip}(g, 10^{-d})] \cdot g$</td>
<td>92.5</td>
<td>92.4</td>
<td>93.8</td>
<td>93.1</td>
</tr>
<tr>
<td>clip($\hat{m}, 10^{-d}$) $\cdot e^\delta$</td>
<td>93.5</td>
<td>92.5</td>
<td>93.8</td>
<td>92.7</td>
</tr>
<tr>
<td>$\hat{m} \cdot e^\delta$</td>
<td>93.1</td>
<td>92.4</td>
<td>93.8</td>
<td>92.6</td>
</tr>
<tr>
<td>$g \cdot e^{\text{sign}(g)\cdot\text{sign}(m)}$</td>
<td>93.1</td>
<td>92.8</td>
<td>93.8</td>
<td>92.8</td>
</tr>
<tr>
<td>drop($g, 0.3$) $\cdot e^{\text{sign}(g)\cdot\text{sign}(m)}$</td>
<td>92.7</td>
<td>92.2</td>
<td>93.6</td>
<td>92.7</td>
</tr>
<tr>
<td>$\hat{m} \cdot e^{g^2}$</td>
<td>93.1</td>
<td>92.5</td>
<td>93.6</td>
<td>92.4</td>
</tr>
<tr>
<td>drop($\hat{m}, 0.1$) / ($e^{g^2} + e$)</td>
<td>92.6</td>
<td>92.4</td>
<td>93.5</td>
<td>93.0</td>
</tr>
<tr>
<td>drop($g, 0.1$) $\cdot e^{\text{sign}(g)\cdot\text{sign}(m)}$</td>
<td>92.8</td>
<td>92.4</td>
<td>93.5</td>
<td>92.2</td>
</tr>
<tr>
<td>clip(RMSProp, $10^{-5}$) + drop($\hat{m}, 0.3$)</td>
<td>90.8</td>
<td>90.8</td>
<td>91.4</td>
<td>90.9</td>
</tr>
<tr>
<td>ADAM * $e^{\text{sign}(g)\cdot\text{sign}(m)}$</td>
<td>92.6</td>
<td>92.0</td>
<td>93.4</td>
<td>92.0</td>
</tr>
<tr>
<td>ADAM * $e^{\hat{m}}$</td>
<td>92.9</td>
<td>92.8</td>
<td>93.3</td>
<td>92.7</td>
</tr>
<tr>
<td>$g + \text{drop}(\hat{m}, 0.3)$</td>
<td>93.4</td>
<td>92.9</td>
<td>93.7</td>
<td>92.9</td>
</tr>
<tr>
<td>drop($\hat{m}, 0.1$) $\cdot e^{g^3}$</td>
<td>92.8</td>
<td>92.7</td>
<td>93.7</td>
<td>92.8</td>
</tr>
<tr>
<td>$g - \text{clip}(g^2, 10^{-4})$</td>
<td>93.4</td>
<td>92.8</td>
<td>93.7</td>
<td>92.8</td>
</tr>
<tr>
<td>$e^g - e^{\hat{m}}$</td>
<td>93.2</td>
<td>92.5</td>
<td>93.5</td>
<td>93.1</td>
</tr>
<tr>
<td>drop($\hat{m}, 0.3$) $\cdot e^{10^{-3}}_w$</td>
<td>93.2</td>
<td>93.0</td>
<td>93.5</td>
<td>93.2</td>
</tr>
</tbody>
</table>

*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 $g \cdot \exp(\text{sign}(g) \cdot \text{sign}(m))$ 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!