Learning to learn by gradient descent by gradient descent. ①

Summary

1. They implement an optimization algorithm with a LSTM.
2. The learned optimizers outperform hand-designed opt. alg. on the tasks they were trained on.
3. The learned optimizers also generalize to new tasks with similar structure.

Learning to learn with RNN

1. Gradient descent takes the form: $\theta_{t+1} = \theta_t - \alpha_t \nabla f(\theta_t)$
2. They consider learning an update rule, g with parameters ϕ.
3. The update to the optimizee f is:
$$\theta_{t+1} = \theta_t + g_t(\nabla f(\theta_t), \phi)$$
4. They write the final optimizee parameters $\theta^*(f, \phi)$ as a fun. of the optimizer parameters ϕ and the function f.
5. One possible obj function for g is: $\mathcal{L}(\phi) = \underset{f}{E}\left[f(\theta^*(f,\phi))\right]$
6. This objective only considers the final optimizee parameters.
7. Another obj function that depends on the entire optimization trajectory is:
$$\mathcal{L}(\phi) = \underset{f}{E}\left[\sum_{t=1}^{T} w_t f(\theta_t)\right]$$
8. where
   - $\theta_{t+1} = \theta_t + g_t$
   - $\begin{bmatrix} g_t \\ h_{t+1} \end{bmatrix} = m(\nabla_t, h_t, \phi)$
   - $w_t \in R_{\geq 0}$ are arbitrary weights
   - g_t : update step, the output of a RNN m.
   - h_t : state of the RNN.
   - T : optimization horizon
   - $\nabla_t = \nabla_\theta f(\theta_t)$

9. They can minimize the value of $\mathcal{L}(\phi)$ with gradient descent on ϕ.

10. They make the assumption that the gradients of the optimizee does not depend on the optimizers parameters.

11. This assumption allows them to avoid computing second derivatives of f.

12. The computational graph can be seen as:

- $\theta_1 \rightarrow \nabla_1 = \nabla_\theta f(\theta_1) \rightarrow \begin{bmatrix} g_1 \\ h_2 \end{bmatrix} = m(\nabla_1, h_1, \phi) \rightarrow \theta_2 = \theta_1 + g_1$

$$\downarrow$$

$$\begin{bmatrix} g_2 \\ h_3 \end{bmatrix} = m(\nabla_2, h_2, \phi)$$

$$\cdots$$

- $\dfrac{\partial \mathcal{L}(\phi)}{\partial \phi} \approx \nabla_\phi \sum_{t=1}^{T} w_t f(\theta_t)$

Coordinatewise LSTM optimizer

1. Using the formulation above, the RNN's hidden state needs to have the same size as the number of parameters of f.

2. This would lead to a huge no. of parameters.

3. To avoid this problem, they use an optimizer m which operates coordinate wise on the parameters of the obj function.

4. This coordinatewise network architecture allows them to use a very small network:
   - only looks at a single coordinate to define the optimizer.
   - share optimizer parameters across different parameters of the optimizee.

5. They argue that the use of recurrence allows the LSTM to learn dynamic update rule which integrates information from the history of gradients, similar to momentum.

6. They use a relatively small LSTM, only 20 hidden units in each layer, 2 layer LSTM.

Experiments on quadratic functions

1. Minimize $f(\theta) = \| W\theta - y \|_2^2$. $W \in R^{10 \times 10}$. $y \in R^{10}$. drawn from IID Gaussian.

2. The optimizers were trained by optimizing random functions from this family and test on newly sampled functions from the same dist.

3. The learned optimizer is uniformly better than the standard optimizers across the no. of steps taken to update the optimizee.

4. They also have experiments on training non-trivially sized NN and demonstrate the benefit of the proposed approach there.

Qs

1. Why is this non-mainstream yet? Is it computationally very expensive.

2.