FINAL PRESENTATION

Daniel Chen, Daniel Savidge, Isaac Bannerman, Jesse Huang

Introduction

- Gradient Descent: The Ultimate Optimizer
    - Kartik Chandra, Audrey Xie, Jonathan Ragan-Kelley, Erik Meijer
- NeurIPS 2022
- https://arxiv.org/pdf/1909.13371.pdf

The Problem

- Hyperparameter search for optimizers for machine learning
  - Important for stability and convergence of neural networks
- Until recently, no automatic way of finding optimal hyperparameters
- This process is expensive
- Manually differentiating update rules is tedious and slow
- Only tunes one hyperparameter at a time
- Introduces a new hyperparameter, hyper-step-size

Paper's Solution

- Automatic Differentiation
  - Automatically computes correct derivatives
  - Naturally generalizes to other hyperparameters
  - Applies to hyperparameters recursively (hyper-hyperparameters, hyper-hyper-hyperparameters, etc.)

Paper's Solution

- Hyperparameter search through reverse mode automatic differentiation (AD)
- For example, with the learning rate hyperparameter alpha

$$\alpha_{i+1} = \alpha_i - k\frac{\partial f(w_i)}{\partial \alpha_i}$$
$$w_{i+1} = w_i - \alpha_{i+1}\frac{\partial f(w_i)}{\partial w_i}$$

- k term can be learned in similar manner, can chain indefinitely

$$k_{i+1} = k_i - k'\frac{\partial f(w_i)}{\partial k_i}$$
$$\alpha_{i+1} = \alpha_i - k_{i+1}\frac{\partial f(w_i)}{\partial \alpha_i}$$

Paper’s Solution

- Automatic differentiation implementation overview with PyTorch
- top: without automatic differentiation, bottom: with automatic differentiation
- Invocation of detach - detachment of variable from computational graph

```
def SGD.__init__(self, alpha):
  self.alpha = alpha

def SGD.step(w):
  d_w = w.grad.detach()
  w = w.detach() - self.alpha.detach() * d_w
```

```
def HyperSGD.step(w):
  # update alpha using equation (1)
  d_alpha = self.alpha.grad.detach()
  self.alpha = self.alpha.detach() - kappa.detach() * d_alpha

  # update w using equation (2)
  d_w = w.grad.detach()
  w = w.detach() - self.alpha.detach() * d_w
```

Paper’s Solution

- Can chain optimizers into stacks
- Left-most: primary optimizer, subsequent: secondary optimizer(s)
- Examples:
    - Adam / SGD
    - RMSProp / SGD
    - SGD / SGD
    - AdaGrad / SGD
    - SGD / SGD / SGD
    - ...

Paper’s Solution

- Outline of implementation of SGD / SGD

```
def HyperSGD.__init__(self, alpha, opt):
  self.alpha = alpha
  self.optimizer = opt

def HyperSGD.step(w):
  self.optimizer.step(self.alpha)
  d_w = w.grad.detach()
  w = w.detach() - self.alpha * d_w

opt = HyperSGD(0.01, opt=SGD(kappa))
```

Results from paper's solution on MNIST

- Test error of neural network with various optimizer stacks compared to baseline
- Baseline: only single primary optimizer in stack, no optimization of hyperparameters
- Experimental design:
  - Fully-connected neural network with a 128-neuron hidden layer with various optimizer stacks
  - Batch size of 256
  - Automatic differentiation method
  - 5 epochs for Adam or RMSProp as primary optimizer, 30 epochs for SGD or AdaGrad as primary optimizer

Results from paper’s solution on MNIST

- Improvement over baseline
- Robust to hyperparameter values to a certain extent

Optimizer	Test error
SGD	8.99±0.05%
SGD / SGD	4.81±0.10%
SGD(0.0769)	5.44±0.10%
SGD / Adam(0.1)	4.86±0.06%
SGD(0.4538)	2.80±0.09%
SGD / AdaGrad	4.85±0.21%
SGD(0.0836)	5.17±0.03%
SGD / RMSprop(0.1)	4.52±0.02%
SGD(0.5920)	2.52±0.07%

(a) Experiments with SGD (Section 3.1)

Optimizer	Test error
Adam	4.67±0.06%
Adam / SGD(10^{-5})	3.03±0.02%
Adam(0.0040, 0.899, 0.999)	3.11±0.06%
Adam$^{\alpha}$ / SGD(10^{-5})	3.12±0.04%
Adam$^{\alpha}$(0.0021)	3.47±0.02%
Adam / Adam	3.05±0.09%
Adam(0.0038, 0.870, 0.999)	3.24±0.13%
Adam$^{\alpha}$ / Adam	3.04±0.08%
Adam$^{\alpha}$(0.0036)	3.08±0.12%

(b) Experiments with Adam (Section 3.2)

Optimizer	Test error
AdaGrad	7.40±0.08%
AdaGrad / SGD	6.90±0.16%
AdaGrad(0.0080)	7.75±0.02%
AdaGrad / AdaGrad	5.03±0.23%
AdaGrad(0.0151)	6.67±0.08%

(c) Experiments with AdaGrad (Section 3.2)

Optimizer	Test error
RMSProp	4.19±0.47%
RMSProp$^{\alpha}$ / SGD(10^{-4})	3.55±0.23%
RMSProp(0.0030)	3.93±0.70%
RMSprop$^{\alpha,\gamma}$ / SGD(10^{-4})	3.33±0.07%
RMSProp(0.0032, 0.9899)	3.25±0.09%
RMSProp$^{\alpha}$ / RMSProp(10^{-4})	3.42±0.45%
RMSProp(0.0021)	3.60±0.04%
RMSProp$^{\alpha,\gamma}$ / RMSProp(10^{-4})	2.96±0.11%
RMSProp(0.0020, 0.9962)	3.65±0.36%

(d) Experiments with RMSProp (Section 3.2)

Cost of paper's solution

- O(1): approximately 1-2% in additional execution time per additional optimizer
- Prior works suggested the differentiation by hand, not automatic

Caveats of paper’s solution

- Rightmost optimizer of stack requires hyperparameter selection
- Large initial hyperparameter, such as large learning rate, can be selected for rightmost optimizer, which can destabilize the stack
  - Can result in large gradients
- Computational graph management

Proposed solution to a caveat

- Gradient clipping: to solve destabilizing effect of large initial learning rate for various optimizer stacks
- Used norm-based gradient clipping as outlined in On the Difficulty of Training Recurrent Neural Networks (Pascanu et al. 2013) paper

Proposed solution to a caveat

For the t-th step across a minibatch:

$$t = 1: \;\; \theta_t \leftarrow ||\mathbf{g}_t||_2$$

$$t > 1: \;\; \theta_t \leftarrow \frac{(\theta_{t-1} * (t-1)) + ||\mathbf{g}_t||_2}{t}$$

For any t:

$$\mathbf{g}'_t \leftarrow \mathbf{g}_t * \min\left(1, \frac{\theta_t}{||\mathbf{g}_t||_2}\right)$$

$$\mathbf{g}_t \leftarrow \mathbf{g}'_t$$

$$\theta'_t \leftarrow \frac{(\theta_{t-1} * (t-1)) + ||\mathbf{g}'_t||_2}{t}$$

$$\theta_t \leftarrow \theta'_t$$

Results of our proposed solution (MNIST)

- Same experimental design as with MNIST dataset in paper, except baseline is no gradient clipping

SGD / SGD(α)		
α	average test error, no gradient clipping	average test error, gradient clipping
0.01	4.90%	5.06%
0.05	6.18%	6.04%
0.1	6.97%	6.68%
0.5	7.01%	7.27%
1	6.78%	6.91%
5	5.31%	5.97%
10	4.72%	5.20%
50	**7.81%**	**3.91%**
75	**38.05%**	**15.92%**
100	**25.26%**	**13.01%**
250	22.75%	40.09%
500	70.28%	72.26%
1000	**80.43%**	**58.04%**

Table 1: Effect of gradient clipping with the SGD / SGD(α) stack on average test error for the MNIST task. Baseline with SGD alone resulted in 8.96% average test error. The bolded rows are where gradient clipping reduced average test error by at least 1%.

Results of our proposed solution (MNIST)

- Same experimental design as with MNIST dataset in paper, except baseline is no gradient clipping

Adam / SGD(α)		
α	average test error, no gradient clipping	average test error, gradient clipping
10^{-5}	3.14%	3.23%
$5 * 10^{-5}$	3.09%	3.21%
10^{-4}	3.05%	3.06%
$5 * 10^{-4}$	3.59%	3.20%
$\mathbf{10^{-3}}$	**4.71%**	**3.35%**
$\mathbf{5 * 10^{-3}}$	**35.89%**	**16.6%**
$\mathbf{10^{-2}}$	**79.57%**	**23.2%**
$\mathbf{5 * 10^{-2}}$	**86.8%**	**84.27%**
10^{-1}	76.17%	86.44%
$5 * 10^{-1}$	83.16%	82.36%

Table 2: Effect of gradient clipping with the Adam / SGD(α) stack on average test error for the MNIST task. Baseline with Adam alone resulted in 4.63% average test error. The bolded rows are where gradient clipping reduced average test error by at least 1%.

Results of our proposed solution (MNIST)

- Same experimental design as with MNIST dataset in paper, except baseline is no gradient clipping

AdaGrad / SGD(α)		
α	average test error, no gradient clipping	average test error, gradient clipping
0.01	6.92%	6.71%
0.05	7.02%	7.16%
0.1	6.75%	7.04%
0.5	5.70%	5.99%
1	**7.86%**	**5.19%**
2.5	**7.34%**	**4.11%**
5	**57.91%**	**3.49%**
7.5	**36.62%**	**30.33%**
10	27.08%	49.9%
50	**64.51%**	**20.79%**
100	**90.19%**	**49.04%**

Table 3: Effect of gradient clipping with the AdaGrad / SGD(α) stack on average test error for the MNIST task. Baseline with AdaGrad alone resulted in 7.38% average test error. The bolded rows are where gradient clipping reduced average test error by at least 1%.

Results of our proposed solution (MNIST)

- Same experimental design as with MNIST dataset in paper, except baseline is no gradient clipping

RMSProp / SGD(α)		
α	average test error, no gradient clipping	average test error, gradient clipping
10^{-5}	4.43%	4.26%
$5 * 10^{-5}$	3.47%	3.84%
10^{-4}	3.36%	3.55%
$5 * 10^{-4}$	4.26%	3.75%
10^{-3}	5.16%	4.86%
$5 * 10^{-3}$	6.16%	6.12%
10^{-2}	6.36%	6.52%
$\mathbf{2.5 * 10^{-2}}$	**88.80%**	**6.14%**
$\mathbf{5 * 10^{-2}}$	**90.66%**	**9.72%**
$7.5 * 10^{-2}$	90.03%	90.03%
$\mathbf{10^{-1}}$	**89.17%**	**64.56%**
$5 * 10^{-1}$	90.2%	90.02%

Table 4: Effect of gradient clipping with the RMSProp / SGD(α) stack on average test error for the MNIST task. Baseline with RMSProp alone resulted in 3.91% average test error. The bolded rows are where gradient clipping reduced average test error by at least 1%.

Results of our proposed solution (MNIST)

- SGD / SGD(alpha):
  - alpha <= ~10, gradient clipping had negligible impact
  - ~10 < alpha <= ~100, gradient clipping reduced test error
- Adam / SGD(alpha):
  - alpha <= ~5 * 10^-4, gradient clipping had negligible impact
  - ~5 * 10^-4 < alpha <= ~5 * 10^-2, gradient clipping reduced test error
- AdaGrad / SGD(alpha):
  - alpha <= ~0.5, gradient clipping had negligible impact
  - ~0.5 < alpha <= ~7.5, gradient clipping reduced test error
- RMSProp / SGD(alpha):
  - alpha <= ~10^-2, gradient clipping had negligible impact
  - ~10^-2 < alpha <= ~5 * 10^-2, gradient clipping reduced test error

Results of our proposed solution (CIFAR-10)

- Same experimental design as with MNIST dataset except neural network is a CNN: two 3 by 3 kernel 16 filter convolutional layers followed by same fully-connected layers as with MNIST

Adam / SGD(α)		
α	average test error, no gradient clipping	average test error, gradient clipping
10^{-9}	39.26%	39.87%
10^{-8}	38.76%	39.12%
10^{-7}	40.02%	39.26%
10^{-6}	**42.17%**	**40.01%**
10^{-5}	**45.95%**	**43.52%**
10^{-4}	**46.96%**	**43.99%**
$2.5 * 10^{-3}$	89.82%	90.49%
$5 * 10^{-3}$	90.00%	90.00%
10^{-3}	**90.00%**	**73.88%**
10^{-2}	90.00%	90.00%

Table 5: Effect of gradient clipping with the Adam / SGD(α) stack on average test error for the CIFAR-10 task. Baseline with Adam alone resulted in 39.65% average test error. The bolded rows are where gradient clipping reduced average test error by at least 1%.

Results of our proposed solution (CIFAR-10)

- Adam / SGD(alpha):
    - alpha <= ~10^-7, gradient clipping had negligible impact
    - ~10^-7 < alpha <= ~10^-4, gradient clipping reduced test error

Cost of our proposed solution

- Gradient clipping calculated in O(1)
- Algorithm still runs in O(1)
- Overall runtime slightly faster due to increased convergence rate and stability

Performance guarantee of our proposed solution

- Hyperparameter size
  - Negligible performance on smaller hyperparameters
  - Mitigated model deterioration on larger hyperparameters
- Equal or better model performance
- No execution time increase

Limitations

- MNIST:
  - Algorithm did not consistently reduce instability for the various two optimizer stacks
- CIFAR-10:
  - Algorithm did not consistently reduce instability in Adam/SGD
  - Inconclusive as to whether gradient clipping with other optimizer stack can reduce test error
- More experiments needed to validate method

Conclusion

- Large hyperparameters caused gradient explosions
  - Solution: Gradient Norm Clipping
- MNIST
  - SGD/SGD(α): $10 < \alpha <= 100$
  - Adam/SGD(α): $10^{-4} < \alpha <= 5*10^{-2}$
  - AdaGrad/SGD(α): $.5 < \alpha <= 7.5$
  - RMSProp/SGD(α): $10^{-2} < \alpha <= 5*10^{-2}$
- CIFAR-10
  - Adam/SGD(α): $10^{-7} < \alpha <= 10^{-4}$
- Small learning rates, gradient clipping had negligible impact
- Large learning rates to an extent, gradient clipping reduced test error