AdaGrad

Gradient Optimizer

David Schulz | Bryan Morales

Computational Problem

- Stochastic gradient descent (SGD) algorithm
  - Scales learning rate with respect to accumulated squared gradient at each iteration
  - Uses the gradients to adjust the learning rate of each parameter
    - **Higher learning rate** -> parameters with **<u>less</u>** frequent features
    - **Smaller learning rate** -> parameters with **<u>more</u>** frequent features

Appropriate for sparse data

Natural Language Processing (NLP)

Image Recognition

Real-World Applications

- **Training a Language Model:**
  - Many words <u>with same meaning</u>
    - Frequency of a word > Frequence of a different word
      - *Learning > Preconditioning*
      - *Use > Utilized*
      - *Close > Nearby*

- **Unsupervised Learning (image recognition):**
  - Model without labeled data
    - Provides information/patterns about dataset
  - Can improve detection of :
    - Human faces
    - Body Parts
    - Different animals

Why use AdaGrad?

- Without AdaGrad:
  - Higher learning rate for common features -> converge quickly
  - Lower learning rate for uncommon features -> can't reach optimal loss

- With AdaGrad:
  - Avoid manually tuning learning rates
  - Infrequent features stand out
  - Model trains each feature at its own pace
  - Helps mitigate distortion in data

Worst & Average time complexity

```
def adagrad(params, states, hyperparams):
    eps = 1e-6
    for p, s in zip(params, states):
        with torch.no_grad():
            s[:] += torch.square(p.grad)
            p[:] -= hyperparams['lr'] * p.grad / torch.sqrt(s + eps)
        p.grad.data.zero_()
```

$$
\begin{aligned}
\mathbf{g}_t &= \partial_{\mathbf{w}} l(y_t, f(\mathbf{x}_t, \mathbf{w})), \\
\mathbf{s}_t &= \mathbf{s}_{t-1} + \mathbf{g}_t^2, \\
\mathbf{w}_t &= \mathbf{w}_{t-1} - \frac{\eta}{\sqrt{\mathbf{s}_t + \epsilon}} \cdot \mathbf{g}_t.
\end{aligned}
$$

- **Worst-Case** time complexity: ***O*** ***(n²)***
  - <u>States:</u> Squared gradients of learning parameters
- **Average-Case** time complexity: ***O*** ***(n²)***

g_t = gradients of parameters
s_t = sum of squared grads
w_t = weights / parameters
η = learning rate

Other useful Algorithms

RMSProp

- Squares previous gradients
- Keeps track of *gamma* variable (leaky average)

```
def rmsprop(params, states, hyperparams):
    gamma, eps = hyperparams['gamma'], 1e-6
    for p, s in zip(params, states):
        s[:] = gamma * s + (1 - gamma) * np.square(p.grad)
        p[:] -= hyperparams['lr'] * p.grad / np.sqrt(s + eps)
```

Worst-Case: ***O*** **(*n*²)** || Average-Case: ***O*** **(*n*²)**

Adadelta

- No master learning rate
- Uses the rate of change of parameters
- Leaky average of the squared rescaled gradients

```
def adadelta(params, states, hyperparams):
    rho, eps = hyperparams['rho'], 1e-5
    for p, (s, delta) in zip(params, states):
        with torch.no_grad():
            # In-place updates via [:]
            s[:] = rho * s + (1 - rho) * torch.square(p.grad)
            g = (torch.sqrt(delta + eps) / torch.sqrt(s + eps)) * p.grad
            p[:] -= g
            delta[:] = rho * delta + (1 - rho) * g * g
        p.grad.data.zero_()
```

Worst-Case: ***O*** **(*n*³)** || Average-Case: ***O*** **(*n*³)**

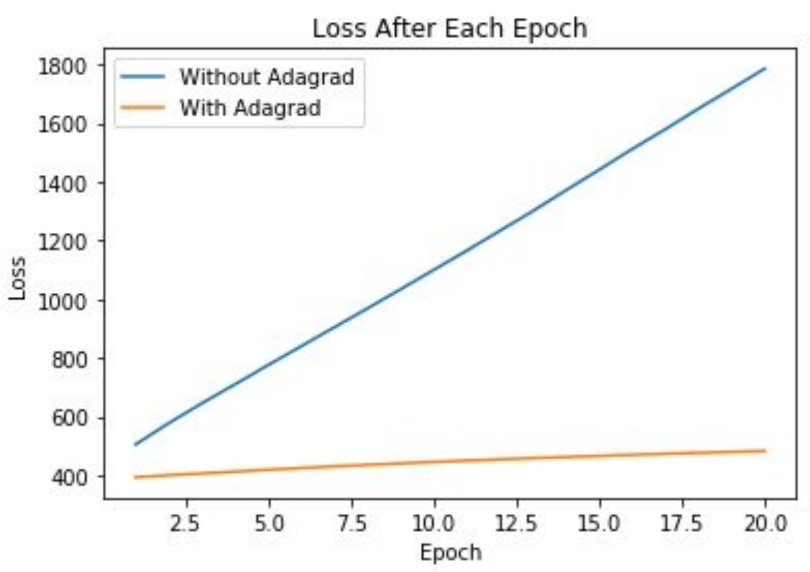
Implementation

```
def init_adagrad_states(parameters):
    list_parameters = []
    for param in parameters:
        list_parameters.append(torch.zeros(param.output.shape))
    return list_parameters


def adagrad(params, states, lr):
    eps = 1e-6
    for p, s in zip(params, states):
        s[:] += torch.square(p.grad)
        p.output -= lr * p.grad / torch.sqrt(s + eps)
        p.clear_grad()
```

```
params = [W,b,c,M]
derivative_list = init_adagrad_states(params)
```

```
# Use gradients to adjust weights
adagrad(params, derivative_list, LEARN_RATE)
network.clear_grad()
# network.step(LEARN_RATE)
```



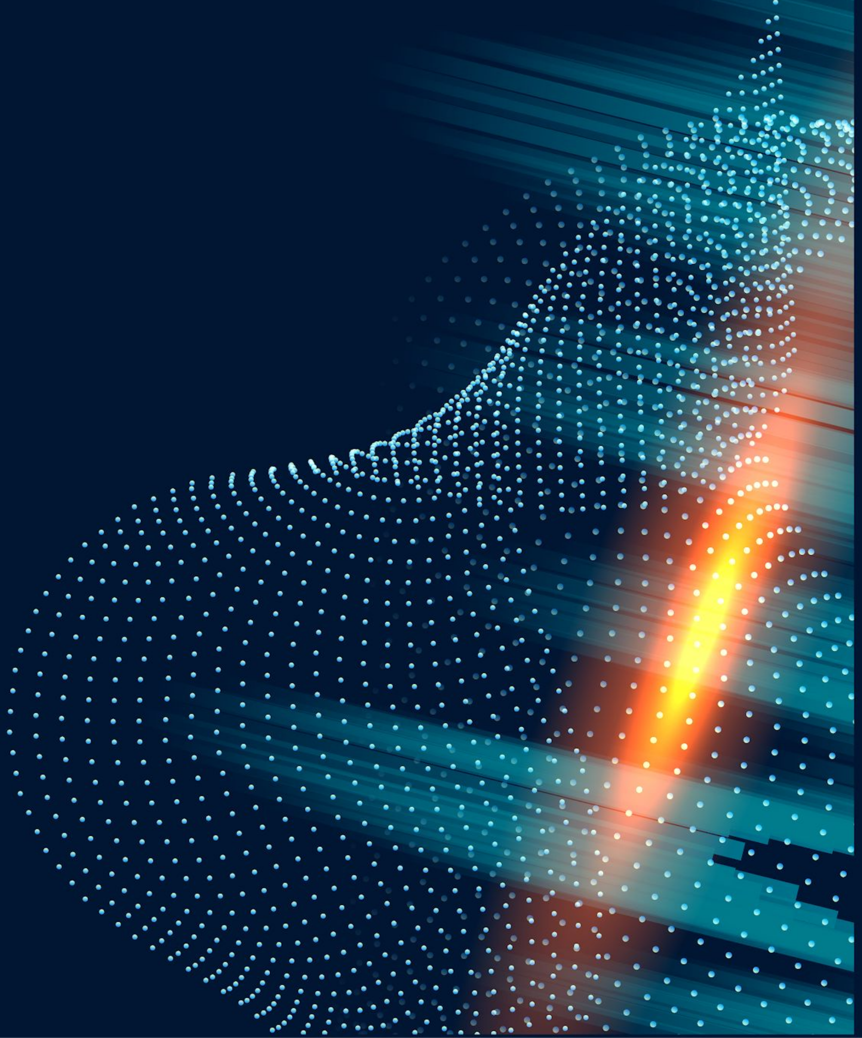
Test Accuracy
Without Adagrad: 0.861
With Adagrad: 0.878

AdaGrad

- One of many optimization algorithms
- Adaptive learning rate for each parameter
- Aids find uncommon features
- Can be too aggressive in reducing learning rates

THANKS!

Any questions?



CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, and infographics & images by Freepik.