Neural Computation

Week 6 - Optimisation Algorithms

Yunwen Lei

School of Computer Science, University of Birmingham

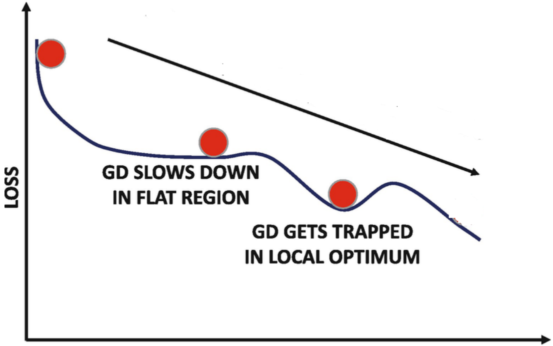
Outline

Optimization with Momentum

Problems of Gradient Descent

Gradient descent can be very slow at flat region (points with small slope)

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla C(\mathbf{w}^{(t)})$$



Around local minima, we are in a flat region!

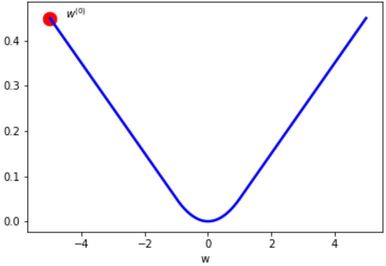
Can we do better?

Yes, Gradient Descent with Momentum!

Gradient Descent with Momentum

Intuition

- If we are repeatedly asked to go in the same direction then we should likely gain some confidence and start taking bigger steps in that direction
- Similar to a ball gaining velocity while rolling down a slope



Gradient Descent with Momentum

- Introduce a velocity to track historic gradients
- $\mathbf{v}^{(0)} = 0$ and update
$$\mathbf{v}^{(t+1)} = \alpha \cdot \mathbf{v}^{(t)} - \eta \nabla C(\mathbf{w}^{(t)})$$
- $\alpha \in [0, 1)$ and update
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{v}^{(t+1)}$$
- In addition to consider current gradient, also look at previous updates

Gradient Descent with Momentum

Velocity is an exponentially decaying moving average of negative gradients

$$\mathbf{v}^{(1)} = \alpha \cdot \mathbf{v}^{(0)} - \eta \cdot \nabla C(\mathbf{w}^{(0)}) = -\eta \cdot \nabla C(\mathbf{w}^{(0)})$$
$$\mathbf{v}^{(2)} = \alpha \cdot \mathbf{v}^{(1)} - \eta \cdot \nabla C(\mathbf{w}^{(1)}) = -\alpha\eta \cdot \nabla C(\mathbf{w}^{(0)}) - \eta \cdot \nabla C(\mathbf{w}^{(1)})$$
$$\vdots$$
$$\mathbf{v}^{(t)} = -\alpha^{t-1}\eta \cdot \nabla C(\mathbf{w}^{(0)}) - \alpha^{t-2}\eta \cdot \nabla C(\mathbf{w}^{(1)}) - \cdots - \eta \cdot \nabla C(\mathbf{w}^{(t-1)})$$

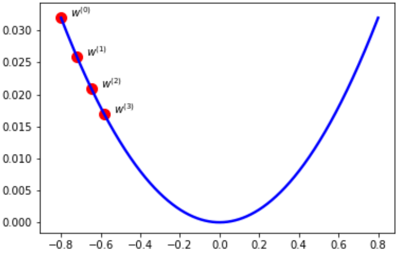
Velocity accumulates previous gradients: larger weights to recent gradients

Two hyperparameters

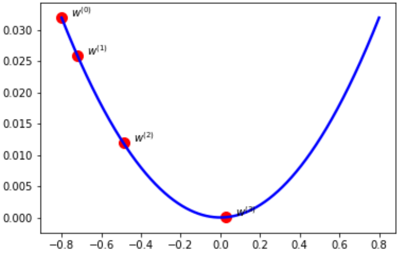
- η: learning rate
- α: determines the influence of past gradients on the current update (usually set as $0.8, 0.9$ or 0.99)

Example: One-dimensional Problem

Consider $C(\mathbf{w}) = \frac{1}{20}\mathbf{w}^2, \mathbf{w} \in \mathbb{R}$. Let $\mathbf{w}^{(0)} = -0.8$. Then $\nabla C(\mathbf{w}) = \frac{1}{10}\mathbf{w}$

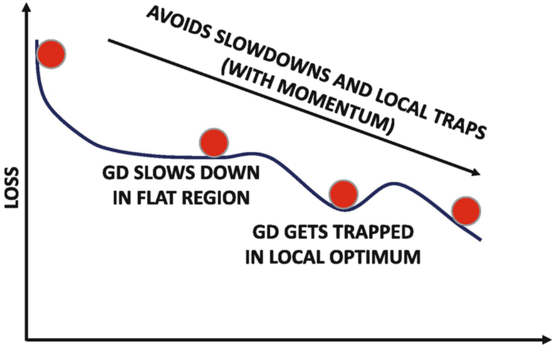


Gradient Descent



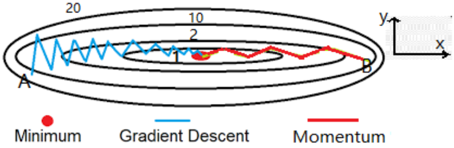
Momentum

Example: One-dimensional Problem



Even in flat regions, the introduction of velocity allows for large steps!

Example: Two-dimensional Problem



x-axis (horizontal direction)

- derivative is < 0 at left side
- derivative is > 0 at right side

y-axis (vertical direction)

- derivative is < 0 at lower side
- derivative is > 0 at upper side

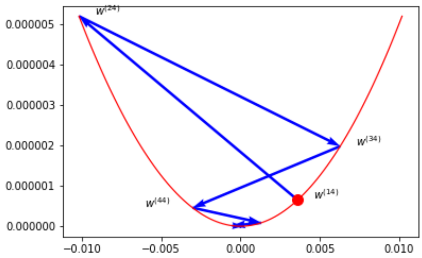
- Each ellipse is a contour line (same function value)
- GD oscillates in vertical direction (derivative in this direction changes sign)
- GD moves consistently in horizontal direction (derivative is small but has same sign)

We wish to accelerate in the horizontal direction and slow down in the vertical direction!

- Momentum damps oscillations in directions of high variation by combining gradients with opposite signs
- It builds up speed in directions with a gentle but consistent gradient

Problems with Momentum

GD with momentum may frequently overshoot minima!



- From $\mathbf{w}^{(0)} = -5$, it moves towards right with increasing speed
- $\mathbf{w}^{(14)}$ is the first iterate moving across optimum
- $\mathbf{w}^{(24)}$ is the first iterate moving across optimum after $\mathbf{w}^{(14)}$
- $\mathbf{w}^{(34)}$ is the first iterate moving across optimum after $\mathbf{w}^{(24)}$
- $\mathbf{w}^{(44)}$ is the first iterate moving across optimum after $\mathbf{w}^{(34)}$ and so on

Nesterov Accelerated Gradient (Nesterov Momentum)

How to reduce the oscillation?

Intuition

- Look ahead before updating
- Recall that for momentum: $\mathbf{v}^{(t+1)} = \alpha \mathbf{v}^{(t)} - \eta \nabla C(\mathbf{w}^{(t)})$
- We are going to move by at least by $\alpha \mathbf{v}^{(t)}$ (known) and a bit more by $\eta \nabla C(\mathbf{w}^{(t)})$ (requires gradient computation)
- Why not virtually moving by $\alpha \mathbf{v}^{(t)}$ to get

$$\mathbf{w}^{(\text{ahead})} = \mathbf{w}^{(t)} + \alpha \mathbf{v}^{(t)}$$

  and use the gradient at $\mathbf{w}^{(\text{ahead})}$
- $\nabla C(\mathbf{w}^{(\text{ahead})})$ may be more precise than $\nabla C(\mathbf{w}^{(t)})$

Nesterov Accelerated Gradient (NAG)

- Look ahead

$$\mathbf{w}^{(\text{ahead})} = \mathbf{w}^{(t)} + \alpha \mathbf{v}^{(t)}$$

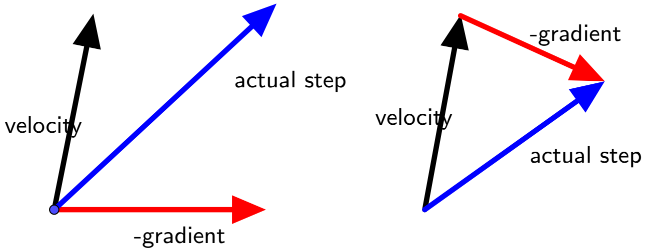
- Calculate gradient and update velocity

$$\mathbf{v}^{(t+1)} = \alpha \mathbf{v}^{(t)} - \eta \nabla C(\mathbf{w}^{(\text{ahead})})$$

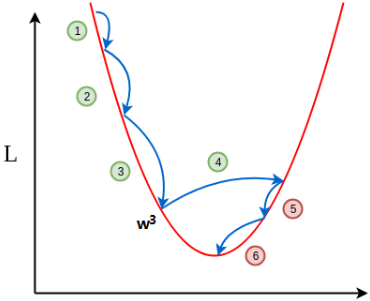
- Update $\mathbf{w}^{(t+1)}$

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{v}^{(t+1)}$$

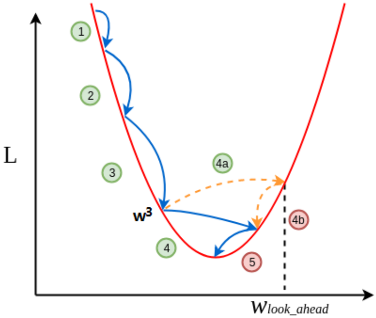
Momentum (Left) versus Nesterov Momentum (Right)



How NAG Relaxes Oscillation?



GD with Momentum



Nesterov Accelerated Gradient

Stochastic Gradient Descent

Stochastic Gradient Descent

Sum Structure

$$C(\mathbf{w}) = \frac{1}{n}\sum_{i=1}^{n} C_i(\mathbf{w}), \quad C_i(\mathbf{w}) \text{ often corresponds to a loss with } i\text{-th example}$$

Stochastic Gradient Descent (Robbins & Monro 1951)

- Initialize the weights $\mathbf{w}^{(0)}$
- For $t = 0, 1, \ldots, T$
  - Draw i_t from $\{1, \ldots, n\}$ with equal probability
  - Compute stochastic gradient $\nabla C_{i_t}(\mathbf{w}^{(t)})$ and update

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \nabla C_{i_t}(\mathbf{w}^{(t)})$$

- computation per iteration is $O(1)$ instead of $O(n)$
- correct on average: if we consider all possible realization of i_t, we recover the true gradient (sum structure)

$$\frac{1}{n}\sum_{i=1}^{n} \nabla C_i(\mathbf{w}^{(t)}) = \nabla C(\mathbf{w}^{(t)})$$

SGD with Momentum

We can combine the idea of SGD and Momentum together!

Algorithm 1: SGD with Momentum

Set initial $\mathbf{w}^{(0)}$ and initial $\mathbf{v}^{(0)} = 0$

for $t = 0, 1, \ldots$ **to** T **do**

  $i_t \leftarrow$ random index from $\{1, 2, \ldots, n\}$ with equal probability

  approximate gradient with selected example $\hat{\mathbf{g}}^{(t)} \leftarrow \nabla C_{i_t}(\mathbf{w}^{(t)})$

  update the velocity $\mathbf{v}^{(t+1)} \leftarrow \alpha \mathbf{v}^{(t)} - \eta \cdot \hat{\mathbf{g}}^{(t)}$

  update the model $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbf{v}^{(t+1)}$

This can be extended to Mini-batch variant!

SGD with Nesterov Momentum

We can combine the idea of SGD and Nesterov Momentum together!

Algorithm 2: SGD with Nesterov Momentum

Set initial $\mathbf{w}^{(0)}$ and initial $\mathbf{v}^{(0)} = 0$
for $t = 0, 1, \ldots$ **to** T **do**
    look ahead $\mathbf{w}^{(\text{ahead})} \leftarrow \mathbf{w}^{(t)} + \alpha \mathbf{v}^{(t)}$
    $i_t \leftarrow$ random index from $\{1, 2, \ldots, n\}$ with equal probability
    approximate gradient with selected example $\hat{\mathbf{g}}^{(\text{ahead})} \leftarrow \nabla C_{i_t}(\mathbf{w}^{(\text{ahead})})$
    update the velocity $\mathbf{v}^{(t+1)} \leftarrow \alpha \mathbf{v}^{(t)} - \eta \cdot \hat{\mathbf{g}}^{(\text{ahead})}$
    update the model $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} + \mathbf{v}^{(t+1)}$

This can be extended to Mini-batch variant!

AdaGrad, RMSProp and Adam

Motivation

Different Features

- $\mathbf{x}_1$: dense, irrelevant
- $\mathbf{x}_2$: sparse, predictive
- $\mathbf{x}_3$: sparse, predictive

	$\mathbf{y}$	$\mathbf{x}_1$	$\mathbf{x}_2$	$\mathbf{x}_3$
$\mathbf{x}^{(1)}$	1	1	0	0
$\mathbf{x}^{(2)}$	-1	.5	0	1
$\mathbf{x}^{(3)}$	1	-.5	1	0
$\mathbf{x}^{(4)}$	-1	0	0	0
$\mathbf{x}^{(5)}$	1	.5	0	0
$\mathbf{x}^{(6)}$	-1	1	0	0
$\mathbf{x}^{(7)}$	1	-1	1	0
$\mathbf{x}^{(8)}$	-1	-.5	0	1

Gradients are multiple of features

- Consider $C_i(\mathbf{w}) = \frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)}\right)^2$
- Gradient becomes

$$\nabla C_i(\mathbf{w}) = \underbrace{(\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)})}_{:=\alpha^{(i)} \in \mathbb{R}} \mathbf{x}^{(i)}$$

Sparse Features Imply Sparse Gradients

Let $\alpha^{(i)} = \mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)}$. Then (we ignore the transpose for space constraint)

$$
\begin{pmatrix} \nabla C_1(\mathbf{w}) \\ \nabla C_2(\mathbf{w}) \\ \vdots \\ \nabla C_n(\mathbf{w}) \end{pmatrix}
=
\begin{pmatrix} \alpha^{(1)}\mathbf{x}^{(1)} \\ \alpha^{(2)}\mathbf{x}^{(2)} \\ \vdots \\ \alpha^{(n)}\mathbf{x}^{(n)} \end{pmatrix}
=
\begin{array}{c}
\begin{array}{ccc} \text{1st coordinate} & \text{2nd coordinate} & \text{3rd coordinate} \end{array} \\
\begin{pmatrix}
\alpha^{(1)} & 0 & 0 \\
0.5\alpha^{(2)} & 0 & \alpha^{(2)} \\
-0.5\alpha^{(3)} & \alpha^{(3)} & 0 \\
0 & 0 & 0 \\
0.5\alpha^{(5)} & 0 & 0 \\
\alpha^{(6)} & 0 & 0 \\
-\alpha^{(7)} & \alpha^{(7)} & 0 \\
-0.5\alpha^{(8)} & 0 & \alpha^{(8)}
\end{pmatrix}
\end{array}
$$

If k-th feature is sparse, then k-th coordinate of gradient is sparse!

Motivation

SGD/GD applies the same step size to all features!

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta_t \nabla C_{i_t}(\mathbf{w}^{(t)}) = \mathbf{w}^{(t)} - \eta_t \begin{pmatrix} \alpha^{(6)} \\ 0 \\ 0 \end{pmatrix} \quad \text{if } i_t = 6$$

Behavior of SGD on sparse and dense features

- SGD will update frequently along dense features
- SGD will scarcely visit examples with non-zero values on sparse features
- SGD will update slowly along sparse features
- This is misleading if dense features are irrelevant and sparse features are relevant

We want to slow down updating on dense features and move fast on sparse features!

AdaGrad (Adaptive Gradient Algorithm)

- Consider different learning rates along different features
- Decay learning rate in proportion to update history (more updates means more decay)

Algorithm 3: AdaGrad (?)

Set initial $\mathbf{w}^{(0)}$ and $\mathbf{r}^{(0)} = (0, \ldots, 0)^\top$

for $t = 0, 1, \ldots$ **to** T **do**

$i_t \leftarrow$ random index from $\{1, 2, \ldots, n\}$ with equal probability

approximate gradient with selected example $\hat{\mathbf{g}}^{(t)} \leftarrow \nabla C_{i_t}(\mathbf{w}^{(t)})$

update the accumulated gradient norm square $\mathbf{r}^{(t+1)} \leftarrow \mathbf{r}^{(t)} + \hat{\mathbf{g}}^{(t)} \odot \hat{\mathbf{g}}^{(t)}$

update the model $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \dfrac{\eta}{\delta + \sqrt{\mathbf{r}^{(t+1)}}} \odot \hat{\mathbf{g}}^{(t)}$

δ is a hyperparameter, often chosen as 10^{-6}

$$\mathbf{r}^{(t+1)} \leftarrow \mathbf{r}^{(t)} + \hat{\mathbf{g}}^{(t)} \odot \hat{\mathbf{g}}^{(t)} \Longleftrightarrow \begin{pmatrix} r_1^{(t+1)} \\ r_2^{(t+1)} \\ \vdots \end{pmatrix} \leftarrow \begin{pmatrix} r_1^{(t)} + (\hat{g}_1^{(t)})^2 \\ r_2^{(t)} + (\hat{g}_2^{(t)})^2 \\ \vdots \end{pmatrix}$$

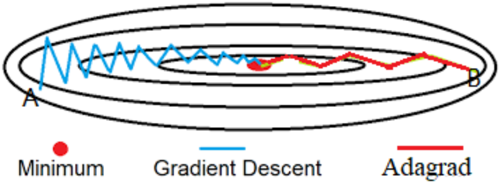
AdaGrad Adjusts Learning Rate in Different Features

$$r_i^{(t+1)} \leftarrow r_i^{(t)} + \left(\hat{g}_i^{(t)}\right)^2$$

- If i-th feature is sparse, then $r_i^{(t)}$ would be small (most $\hat{g}_i^{(t)}$ are 0)
- If i-th feature is dense, then $r_i^{(t)}$ would be large
- By dividing the learning rate with $\sqrt{\mathbf{r}^{(t)}}$, AdaGrad would increase learning rate in sparse features and decrease learning rate in dense features

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \frac{\eta}{\delta + \sqrt{\mathbf{r}^{(t+1)}}} \odot \hat{\mathbf{g}}^{(t)} \Longleftrightarrow \begin{pmatrix} w_1^{(t+1)} \\ w_2^{(t+1)} \\ \vdots \end{pmatrix} \leftarrow \begin{pmatrix} w_1^{(t)} - \frac{\eta g_1^{(t)}}{\delta + \sqrt{r_1^{(t+1)}}} \\ w_2^{(t)} - \frac{\eta g_2^{(t)}}{\delta + \sqrt{r_2^{(t+1)}}} \\ \vdots \end{pmatrix}$$

AdaGrad: Another Interpretation



Want acceleration in horizontal direction and slow down in vertical direction

This can be achieved by AdaGrad $r_i^{(t+1)} \leftarrow r_i^{(t)} + (\hat{g}_i^{(t)})^2$.

- Function changes rapidly (vertical), implying a large derivative (and r) in this direction.
- Function changes slowly (horizontal), implying a small derivative (and r) in this direction

RMSProp (Root Mean Square Propagation)

Intuition

- Adagrad decays learning rate very aggressively (as denominator grows)
- After a while the dense feature will receive small updates
- RMSProp is introduced to prevent rapid growth of denominator ($\beta = 0.9$)

Algorithm 4: RMSProp

Set initial $\mathbf{w}^{(0)}$ and $\mathbf{r}^{(0)} = (0, \dots, 0)^\top$
for $t = 0, 1, \dots$ **to** T **do**
  $i_t \leftarrow$ random index from $\{1, 2, \dots, n\}$ with equal probability
  approximate gradient with selected example $\hat{\mathbf{g}}^{(t)} \leftarrow \nabla C_{i_t}(\mathbf{w}^{(t)})$
  update accumulated gradient norm square $\mathbf{r}^{(t+1)} \leftarrow \beta \cdot \mathbf{r}^{(t)} + (1 - \beta)\hat{\mathbf{g}}^{(t)} \odot \hat{\mathbf{g}}^{(t)}$
  update the model $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \dfrac{\eta}{\delta + \sqrt{\mathbf{r}^{(t+1)}}} \odot \hat{\mathbf{g}}^{(t)}$

This can be extended to Mini-batch variant!

Adam (Adaptive Moment Estimation)

Intuition

- Use the idea of momentum to memorize previous gradients
- Use the idea of RMSProp to distinguish updates along features

Algorithm 5: Adam(?)

Set initial $\mathbf{w}^{(0)}, \mathbf{r}^{(0)} = 0, \mathbf{s}^{(0)} = 0$

for $t = 0, 1, \ldots$ **to** T **do**

$i_t \leftarrow$ random index from $\{1, 2, \ldots, n\}$ with equal probability

approximate gradient with selected example $\hat{\mathbf{g}}^{(t)} \leftarrow \nabla C_{i_t}(\mathbf{w}^{(t)})$

update the momentum $\mathbf{s}^{(t+1)} \leftarrow \beta_1 \cdot \mathbf{s}^{(t)} + (1 - \beta_1)\hat{\mathbf{g}}^{(t)}$

update accumulated gradient norm square $\mathbf{r}^{(t+1)} \leftarrow \beta_2 \cdot \mathbf{r}^{(t)} + (1 - \beta_2)\hat{\mathbf{g}}^{(t)} \odot \hat{\mathbf{g}}^{(t)}$

bias correction $\hat{\mathbf{s}}^{(t+1)} \leftarrow \mathbf{s}^{(t+1)}/(1 - \beta_1^{t+1}), \quad \hat{\mathbf{r}}^{(t+1)} \leftarrow \mathbf{r}^{(t+1)}/(1 - \beta_2^{t+1})$

update the model $\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \dfrac{\eta}{\delta + \sqrt{\hat{\mathbf{r}}^{(t+1)}}} \odot \hat{\mathbf{s}}^{(t+1)}$

Adam

Hyperparameters (typical choice)

- $\eta = 0.001$
- $\beta_1 = 0.9$
- $\beta_2 = 0.999$
- $\delta = 10^{-8}$

Adam is widely used in deep learning!

Summary

Comparison between GD and SGD

- GD converges faster by iteration counts
- GD requires more computation per iteration

Alternatives of SGD

- Momentum: velocity $\mathbf{v}^{(t+1)} = \alpha \mathbf{v}^{(t)} - \eta \nabla C(\mathbf{w}^{(t)})$
- NAG: look ahead $\mathbf{v}^{(t+1)} = \alpha \mathbf{v}^{(t)} - \eta \nabla C\left(\mathbf{w}^{(t)} + \alpha \mathbf{v}^{(t)}\right)$
- AdaGrad: feature-wise learning rate $\mathbf{r}^{(t+1)} = \mathbf{r}^{(t)} + \hat{\mathbf{g}}^{(t)} \odot \hat{\mathbf{g}}^{(t)}$
- RMSProp: slow down learning rate decay $\mathbf{r}^{(t+1)} = \beta \mathbf{r}^{(t)} + (1-\beta)\hat{\mathbf{g}}^{(t)} \odot \hat{\mathbf{g}}^{(t)}$
- Adam: momentum+RMSProp

$$\mathbf{s}^{(t+1)} = \beta_1 \cdot \mathbf{s}^{(t)} + (1-\beta_1)\hat{\mathbf{g}}^{(t)}, \mathbf{r}^{(t+1)} \leftarrow \beta_2 \cdot \mathbf{r}^{(t)} + (1-\beta_2)\hat{\mathbf{g}}^{(t)} \odot \hat{\mathbf{g}}^{(t)}.$$

How to choose appropriate algorithms still remains open research problems!

References I