Week 6: Optimisation Algorithms

Yunwen Lei

School of Computer Science, University of Birmingham

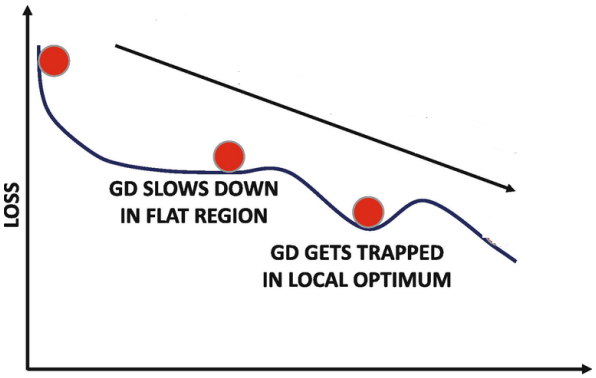
In the previous study, we have introduced the fundamental gradient descent algorithm. In this week, we will introduce more advanced optimization algorithms. These algorithms are widely used in solving large-scale optimization problems in machine learning and data science. For each algorithm, we will introduce both the implementation and the motivation.

1 Optimization with Momentum

1.1 Problems with Gradient Descent

While gradient descent is fundamental and easy to implement, it suffers from some drawbacks. A disadvantage of gradient descent is that it moves very slowly at flat region. By flat region we mean the region where the function values change slowly. This phenomenon can be illustrated by its update formula

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla C(\mathbf{w}^{(t)}) \qquad (1.1)$$



At the initialization, the function value changes very fast and therefore the magnitude of the gradient is large. According to Eq. (1.1), gradient descent updates the iterates fast in the sense that $\|\mathbf{w}^{(t+1)} - \mathbf{w}^{(t)}\|_2$ is large. However, if we get trapped to a flat region, the gradient would be small and then the update would be very slow. This means that we get trapped to that flat region. If we are fortunate to escape this flat region, then we can continue to decrease the function value. However, when we are closed to a local optimum we will get trapped to that local optimum. The reason is that the gradient at the local optimum is zero (by the first-order optimality condition).

This motivates the following important question:

Can we develop a better algorithm to speed up gradient descent?

The answer is yes and the idea is to use velocity (which is also called momentum).

1.2 Gradient Descent with Momentum

We first introduce some 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.

To explain this, assume we want to drive a car to UoB but we do not know the direction. Then we have to ask other people for the direction. Suppose we meet a person and she tells us that we should move

toward the north direction. This is useful information but we are not certain whether this information is accurate. Therefore, we move toward the north direction with a slow speed. After 10 minutes, we ask another person and she tells us that we should move toward the north direction. Then we are more confident that north direction is a correct direction and therefore we can speed up. After another 20 minutes, another person also says that we should move along the north direction. Then we are quite certain and will use larger speed.

This process is similar to a ball gaining velocity while rolling down a slope. In the beginning, we are located at the red point and we have velocity 0. The gravity will drag the ball to the right direction. In this process, the ball is gaining velocity due to the gravity and will move faster and faster.

A key concept in the above example is the velocity, which stores the historical information. Motivated by this phenomenon, we introduce a velocity to track historic gradients.

- In the beginning, we initialize the velocity $\mathbf{v}^{(0)} = 0$. Then we update the velocity as follows

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

  According to this formula, we use both the previous velocity $\mathbf{v}^{(t)}$ and the new information $\nabla C(\mathbf{w}^{(t)})$ to update the velocity. Since previous velocity stores historic information, the new velocity will store both the historic information and the new information.

- $\alpha \in [0, 1)$ and update

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

By the update of velocity, we know that velocity is an exponentially decaying moving average of negative gradients

$$\begin{aligned}
\mathbf{v}^{(1)} &= \alpha \cdot \mathbf{v}^{(0)} - \eta \cdot \nabla C(\mathbf{w}^{(0)}) = -\eta \cdot \nabla C(\mathbf{w}^{(0)}) \\
\mathbf{v}^{(2)} &= \alpha \cdot \mathbf{v}^{(1)} - \eta \cdot \nabla C(\mathbf{w}^{(1)}) = -\alpha\eta \cdot \nabla C(\mathbf{w}^{(0)}) - \eta \cdot \nabla C(\mathbf{w}^{(1)}) \\
\mathbf{v}^{(3)} &= \alpha \cdot \mathbf{v}^{(2)} - \eta \cdot \nabla C(\mathbf{w}^{(2)}) = -\alpha^2\eta \cdot \nabla C(\mathbf{w}^{(0)}) - \alpha\eta \cdot \nabla C(\mathbf{w}^{(1)}) - \eta \cdot \nabla C(\mathbf{w}^{(2)}) \\
&\vdots \\
\mathbf{v}^{(t)} &= -\alpha^{t-1}\eta \cdot \nabla C(\mathbf{w}^{(0)}) - \alpha^{t-2}\eta \cdot \nabla C(\mathbf{w}^{(1)}) - \cdots - \eta \cdot \nabla C(\mathbf{w}^{(t-1)})
\end{aligned}$$

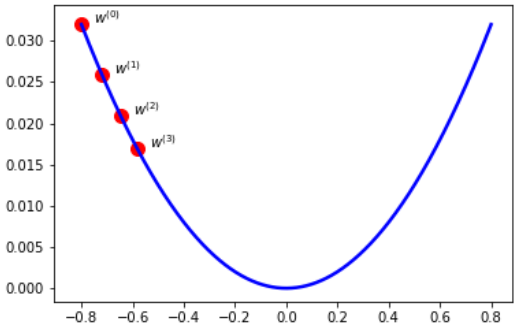
It is then clear that velocity accumulates previous gradients. We give larger weights to recent gradients in the velocity. This is reasonable since the recent gradient is more useful to the current update than the old gradients.

Remark 1 (Hyperparameters)**.** There are two hyperparameters in gradient descent with momentum.

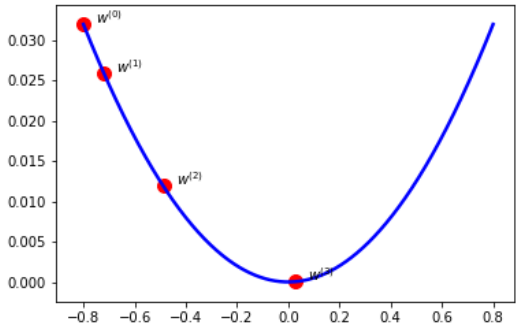
- η: learning rate
- α: determines the influence of past gradients on the current update (usually set as $0.8, 0.9$ or 0.99)

Example 1 (One-dimensional Problem)**.** Now we apply gradient descent (GD) and gradient descent with momentum (Momentum) to minimize a one-dimensional problem $C(\mathbf{w}) = \frac{1}{20}\mathbf{w}^2, \mathbf{w} \in \mathbb{R}$. It is clear that the gradient $\nabla C(\mathbf{w}) = \frac{1}{10}\mathbf{w}$. Suppose we initialize the point $\mathbf{w}^{(0)} = -0.8$. The left figure shows the behavior of GD while the right figure shows the behavior of Momentum.

From these figures we can see that GD updates slower and slower. The underlying reason is that the magnitude of gradient is $|\mathbf{w}|/10$, which is becoming smaller and smaller. After 3 updates GD is still at a point far from the optimum. On the other hand side, Momentum will move faster and faster. The underlying reason is that the magnitude of velocity will become larger and larger since velocity would accumulate the previous gradient information. Within 3 updates, Momentum finds a point very close to the global minimum.



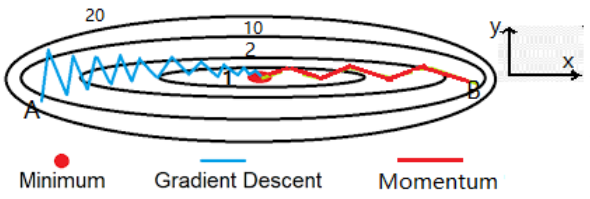
Gradient Descent



Momentum

Example 2 (One-dimensional Problem)**.** Recall that GD has a problem of getting trapped to flat region and local optimum. By using the velocity, Momentum can solve these problems. The underlying reason is that even in flat region with a very small gradient, the velocity can still have a large magnitude. This velocity will allow us to escape this flat region and a local optimum!

Example 3 (Two-dimensional Problem)**.** We now show the performance of Momentum to a two-dimensional problem. The figure shows a contour plot of the function we want to minimize. Each ellipse is a contour, meaning that the function value is the same for all points in the same ellipse. The minimum is achieved in the red point, which has the function value 1. For ellipses from inside to outside, the function values are 2, 10, and 20 respectively. Each ellipse has a long x axis and a short y axis. This means that if we want the same change of function value, we can either move a short distance in the y axis or a long distance in the x axis. This further shows that the derivative is large in the vertical direction and small in the horizontal direction (recall derivative is the change rate of function value w.r.t. the arguments, the change of the function value is the same, the change of the argument in the vertical direction is small while the change of the argument in the horizontal direction is large).



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

We can further show the behavior of partial derivative. We first consider the partial derivative w.r.t. x. If x is at the left side of the red point, the partial derivative is negative. The reason is that the function value decreases from 20 to 1 when we increase x. If x is at the right side of the red point, the partial derivative is positive. The reason is that the function value increases from 1 to 20 if we increase x.

Now we consider the partial derivative w.r.t. y. Similarly, one can show that the partial derivative is positive if y is at the upper side of the red point, and negative if y is at the lower side of the red point.

The blue line shows the behavior of GD.

- GD oscillates in vertical direction (derivative in this direction changes sign). The reason is that the magnitude of the derivative is large in the vertical direction. Therefore, with a small step size η the update of vertical coordinate can be fast $w_y^{(t+1)} = w_y^{(t)} - \eta \frac{\partial C(w)}{\partial w_y^{(t)}}$.

- GD moves consistently in horizontal direction (derivative is small but has same sign). The reason is that the magnitude of the derivative is small in the horizontal direction. Therefore, with a moderate step size η the update of horizontal coordinate would be slow $w_x^{(t+1)} = w_x^{(t)} - \eta \frac{\partial C(w)}{\partial w_x^{(t)}}$

We wish to accelerate in the horizontal direction and slow down in the vertical direction. This can be achieved by Momentum.

The key reason is that the velocity is a moving summation of negative gradients.

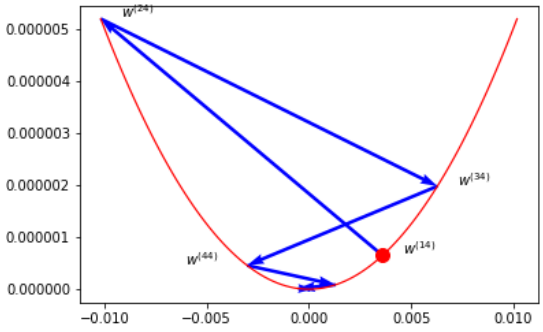
- Momentum damps oscillations in directions of high variation by combining gradients with opposite signs (positive and negative)
- It builds up speed in directions with a gentle but consistent gradient (with the same sign)

In the vertical direction, gradient descent oscillates and the derivative changes sign frequently. Therefore, by taking a moving summation of previous derivatives, the positive and negative derivatives will be cancelled off and this smooth the implementation of gradient descent in the vertical direction. In the horizontal direction, gradient descent moves slowly and the derivative in the optimization process have the same sign. By taking a moving summation of previous derivatives, the magnitude of velocity will become larger and larger. Then we accelerate the optimization in the horizontal direction.

This shows the magic of Momentum: it speeds up the optimization in horizontal direction, and smoothes the optimization in the vertical direction. The behavior is illustrated in the red line.

1.3 Nesterov Accelerated Gradient (Nesterov Momentum)

A problem with Momentum is that it may frequently overshoot minima! This can be illustrated by the following example where the minimum is achieved at the zero point. We start at the point $\mathbf{w}^{(0)} = -5$. It moves towards right with increasing speed. The figure shows the iterate $\mathbf{w}^{(t)}$.



- $\mathbf{w}^{(14)}$ 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

According to the above figure, we know that these iterates would fluctuate around the global minimum. The reason is that the magnitude of the velocity will increase quickly until it goes over the optimum. To avoid this problem, we need to slow down the increasing speed of the velocity.

An algorithm to reduce the oscillation of Momentum is Nesterov Accelerated Gradient (Nesterov Momentum). The intuition is to look ahead before updating.

- Recall that for Momentum: $\mathbf{v}^{(t+1)} = \alpha\mathbf{v}^{(t)} - \eta\nabla C(\mathbf{w}^{(t)})$
- According to Momentum, we are going to move by at least by $\alpha\mathbf{v}^{(t)}$ and a bit more by $\eta\nabla C(\mathbf{w}^{(t)})$. The term $\alpha\mathbf{v}^{(t)}$ is already (known), while the term $\nabla C(\mathbf{w}^{(t)})$ (requires gradient computation).
- Since we already know the term $\alpha\mathbf{v}^{(t)}$, why not virtually moving by $\alpha\mathbf{v}^{(t)}$ to get

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

  and do the gradient computation at the point $\mathbf{w}^{(\text{ahead})}$.

- This has the same computation cost as the Momentum. However, $\nabla C(\mathbf{w}^{(\text{ahead})})$ may be more precise than $\nabla C(\mathbf{w}^{(t)})$ since the point $\mathbf{w}^{(\text{ahead})}$ can be a better point than $\mathbf{w}^{(t)}$ (note we make a better use of the velocity in getting $\mathbf{w}^{(\text{ahead})}$).

This leads to the famous algorithm Nesterov Accelerated Gradient.

Definition 1 (Nesterov Accelerated Gradient (NAG))**.**

- Look ahead

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

- Calculate gradient and update velocity

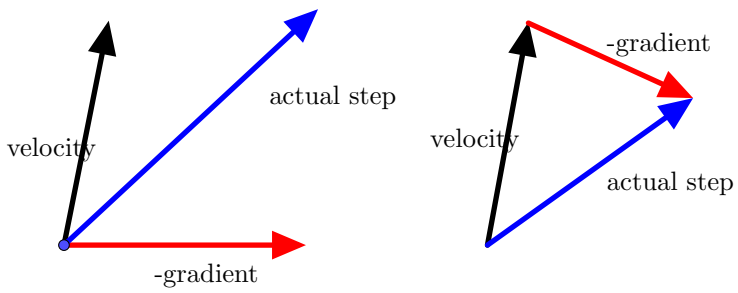
$$\mathbf{v}^{(t+1)} = \alpha \mathbf{v}^{(t)} - \eta \nabla C(\mathbf{w}^{(\text{ahead})})$$

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

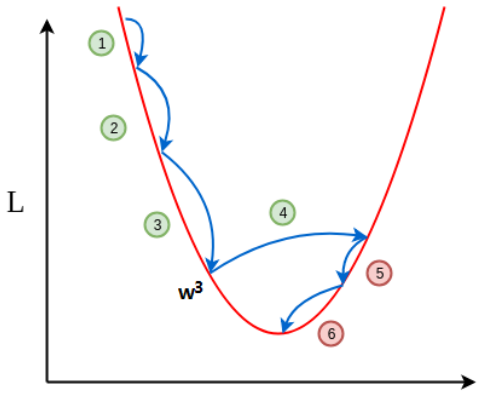
$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} + \mathbf{v}^{(t+1)}$$

The only difference between NAG and Momentum is that we choose a different point in the gradient computation. The gradient computation is the most expensive part in the gradient-based methods. Therefore, we need to be very careful in choosing the point for gradient computation. The following figure visualize the difference between Momentum and NAG

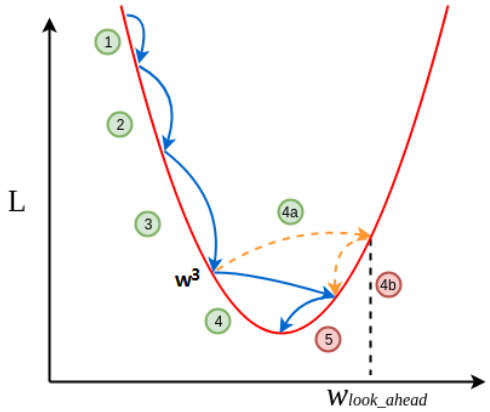
Momentum (Left) versus Nesterov Momentum (Right)



Example 4 (How NAG Relaxes Oscillation?)**.** We now give an example to show how NAG relaxes the oscillation of Momentum. The left figure shows that GD with momentum overshoots the global minimum at $\mathbf{w}^{(3)}$ due to a large velocity. We now consider NAG at this point. It is clear the derivative at $\mathbf{w}^{(3)}$ is negative since the function value is decreasing at this point. The velocity is positive. According to NAG, we first build a look-ahead point. This look-ahead already overshoots the global minimum and the derivative at this look-ahead point is positive. Then we update the velocity (which is positive) by adding the negative gradient, and therefore we get a new velocity with a smaller magnitude. This shows that we move with a slower speed at $\mathbf{w}^{(3)}$ to get $\mathbf{w}^{(4)}$. Although $\mathbf{w}^{(4)}$ also overshoots the global minimum, the level of overshooting is smaller than GD with momentum.



GD with Momentum



Nesterov Accelerated Gradient

2 Stochastic Gradient Descent with Momentum

Property 1 (Sum Structure)**.** The function we minimize in machine learning often has a sum structure:

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

GD does not use the sum structure. It just uses the gradient at C to update iterates. This gradient computation requires to go through all the training examples, and can be expensive if n is large. Nowadays we often encounter a data-set of large scale. In this case, GD is no longer efficient. This motivates a key question

Can we use the sum structure to develop a more efficient method?

A very famous algorithm to this aim is the stochastic gradient descent (SGD). The key idea is to estimate the true gradient $\nabla C(\mathbf{w}_t)$ based on a randomly selected example.

Definition 2. 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

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

Remark 2.

- We say $\nabla C_{i_t}(\mathbf{w}^{(t)})$ a stochastic gradient, which is a random variable. Since we only use a single example to build a stochastic gradient, the computation per iteration is $O(1)$ instead of $O(n)$. This greatly relaxes the computation burden
- While this stochastic gradient is not a correct gradient, it is correct on average: if we consider all possible realization of i_t, we recover the true gradient (sum structure)

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

Note i_t has the same likelihood to be any number in $\{1, 2, \ldots, n\}$. The term in the left-hand side is the expectation of the stochastic gradient. This shows that on-average, the stochastic gradient is a good estimation of the true gradient. This explains why we can use the stochastic gradient in the iterate update. Indeed, people have already theoretically proved the SGD can find a minimizer of the problem under some assumptions.

Extension 1 (SGD with Momentum)**.** We can combine the idea of SGD and the idea of momentum together. This leads to SGD with momentum. This is similar to GD with momentum except that we replace the gradient by a stochastic gradient computed on a randomly sampled example.

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

Extension 2 (SGD with Nesterov Momentum)**.** We can combine the idea of SGD and the idea of Nesterov momentum together. This leads to SGD with Nesterov momentum. This is similar to NAG except that we replace the gradient by a stochastic gradient computed on a randomly sampled example.

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

3 AdaGrad, RMSProp and Adam

Learning rate is an important parameter which has huge influence on the performance of the algorithm. In the previous part, the step size is not-adaptive to the dataset, meaning that we do not adjust the learning rate according to the dataset. In this part, we will introduce adaptive gradient methods. We mainly discuss three methods: AdaGrad, RMSProp and Adam.

3.1 AdaGrad

Before introducing AdaGrad, we first give some motivation. Suppose we consider a binary classification problem on a dataset with three features. The details of the dataset are as follows

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

Here the features have different property (we use sub-index to differentiate features and super-index to differentiate examples)

- The feature x_1 is a dense feature and is irrelevant to the prediction of the label y. By irrelevant we mean that when x_1 takes the same value the label y can have different values. For example, we have $x_1 = 0.5$ for the second example and the corresponding label is -1. We have $x_1 = 0.5$ in the fifth example and the corresponding label is 1.
- The feature x_2 is a sparse feature and is predictive. By sparsity we mean that most of examples have 0 in this feature. By predictive we mean that if $x_2 = 1$, the corresponding label is always 1.
- In a similar way, one can show that x_3 is a sparse and predictive feature.

Suppose we apply SGD to this problem. To understand the behavior we first compute the gradient.

Property 2 (Gradients are multiple of features)**.**

- The loss of a model $\mathbf{w}$ on the i-th example is $C_i(\mathbf{w}) = \frac{1}{2}\left(\mathbf{w}^\top \mathbf{x}^{(i)} - y^{(i)}\right)^2$
- The gradient becomes

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

  According to the above equation, we know that the gradient can be represented by $\mathbf{x}^{(i)}$ multiplied by a real number.

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

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

A key observation is that

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

We now use the above observation to analyze the performance of SGD. Notice that sparse features mean sparse coordinates of gradients. Now we are interested in the second feature x_2. This is a sparse feature, which means most of examples have 0 on this feature. Therefore, for most of iterates SGD will select an example with 0 on this feature. This means that SGD will not update the second coordinate $\mathbf{w}_2$, i.e., $\mathbf{w}_2^{(t+1)} = \mathbf{w}_2^{(t)}$. For example, if we have $i_t = 6$ then

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

Then SGD make scarce updates on the second coordinate. However, we know that x_2 is a predictive feature. We wish effective updates on the corresponding coordinate. We summarize this behavior as follows

- 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! This can be achieved by AdaGrad (Adaptive Gradient Algorithm). The idea is to introduce accumulated gradient norm square $\mathbf{r}^{(t+1)}$ as follows

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

where δ is a hyperparameter (often chosen as 10^{-6}). We introduce δ to make sure that the denominator is not zero. The step size along the i-th feature is then $\eta/(\delta + \sqrt{r_i^{(t+1)}})$.

- Since $r_i^{(t+1)}$ is different for different i, we have different learning rates along different features.
- Decay learning rate in proportion to update history. We give more decays to a feature if there is already a lot of update on that feature. Frequent updates means that the accumulated gradient norm square is large, which further means the step size is small. This achieves the effect of slowing down the update. Furthermore, if a feature is updated scarcely then the corresponding accumulated gradient norm square is small, which further means the step size is large on the feature. The achieves the effect of speeding up the update.

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

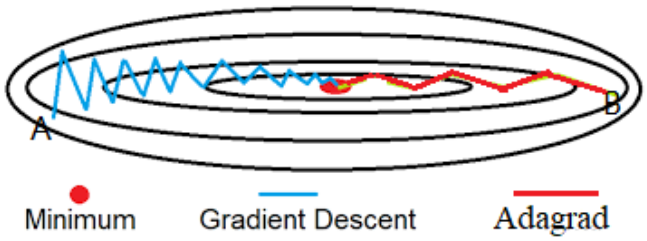
Remark 3 (AdaGrad Adjusts Learning Rate in Different Features)**.** The update on the i-th feature is

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

- 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

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

Example 5 (Another Interpretation of AdaGrad)**.** Let us consider the minimization on this problem



According to Example 3, we want acceleration in horizontal direction and slow down in vertical direction. We show before that this can be achieved by Momentum. We now show that we can achieve this by AdaGrad $r_i^{(t+1)} \leftarrow r_i^{(t)} + (\hat{g}_i^{(t)})^2$.

- According to Example 3, we have a large derivative in the y-axis. This means that the accumulated gradient norm square is large in this feature, which means the learning rate is small. Therefore, this slows down the update in the vertical direction.
- We have a small derivative in the x-axis. This means that the accumulated gradient norm square is small in this feature, which means the learning rate is large. Therefore, this speeds up the update in the horizontal direction.

3.2 RMSProp

RMSProp (Root Mean Square Propagation) is a simple variant of AdaGrad. The intuition is as follows

- Adagrad decays learning rate very aggressively

$$\mathbf{r}^{(t+1)} \leftarrow \mathbf{r}^{(t)} + \hat{\mathbf{g}}^{(t)} \odot \hat{\mathbf{g}}^{(t)}.$$

- Since $\mathbf{r}^{(t+1)}$ continues to increase, the learning rate is becoming smaller and smaller. After a while the dense feature will receive small updates. This slows down the optimization process
- RMSProp is introduced to prevent rapid growth of denominator. The key idea is to update the accumulated gradient norm square as follows

$$\mathbf{r}^{(t+1)} \leftarrow \beta \cdot \mathbf{r}^{(t)} + (1-\beta)\hat{\mathbf{g}}^{(t)} \odot \hat{\mathbf{g}}^{(t)},$$

  where β is a hyperparameter (a typical choice is $\beta = 0.9$). In this way, $\mathbf{r}^{(t+1)}$ grows much more slowly and we can still have effective updates after many iterations.

The detailed algorithm is as follows.

Algorithm 4 RMSProp

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

3.3 Adam

Finally, we introduce Adam, which is a most popular optimization algorithm in training deep neural networks. Notice that momentum introduces a velocity to keep the historical information on the gradients. AdaGrad introduces a accumulated gradient norm square to give different learning rates for different features. This motivates the question

Can we combine the idea of both algorithms to develop an efficient algorithm?

This leads to Adam implemented as follows

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

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

end

Adam uses the idea of momentum to memorize previous gradients, and uses the idea of RMSProp to distinguish updates along features. Therefore, we can expect it inherit the advantage of both algorithms. Notice there is a bias correction step. The interpretation is very tricky and we do not give explanations here. If you are interested, you can check the link `https://www.youtube.com/watch?v=lWzo8CajF5s&t=178s` for details (we will not cover the interpretation of bias correction in the exam). There are four hyperparameters in Adam. Here are some typical choices: $\eta = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\delta = 10^{-8}$.