 [Chart.js](http://docs.google.com/docs/3.9.1/)

[Home](http://docs.google.com/docs/3.9.1/)

[API](http://docs.google.com/docs/3.9.1/api/)

[Samples](http://docs.google.com/docs/3.9.1/samples/)

Ecosystem Ecosystem

* [Awesome (opens new window)](https://github.com/chartjs/awesome)
* [Slack (opens new window)](https://chartjs-slack.herokuapp.com/)
* [Stack Overflow (opens new window)](https://stackoverflow.com/questions/tagged/chart.js)

[GitHub (opens new window)](https://github.com/chartjs/Chart.js)

[Home](http://docs.google.com/docs/3.9.1/)

[API](http://docs.google.com/docs/3.9.1/api/)

[Samples](http://docs.google.com/docs/3.9.1/samples/)

Ecosystem Ecosystem

* [Awesome (opens new window)](https://github.com/chartjs/awesome)
* [Slack (opens new window)](https://chartjs-slack.herokuapp.com/)
* [Stack Overflow (opens new window)](https://stackoverflow.com/questions/tagged/chart.js)

[GitHub (opens new window)](https://github.com/chartjs/Chart.js)

* [Chart.js](http://docs.google.com/docs/3.9.1/)
* Getting Started
* General
  + [Accessibility](http://docs.google.com/docs/3.9.1/general/accessibility.html)
  + [Colors](http://docs.google.com/docs/3.9.1/general/colors.html)
  + [Data structures](http://docs.google.com/docs/3.9.1/general/data-structures.html)
  + [Fonts](http://docs.google.com/docs/3.9.1/general/fonts.html)
  + [Options](http://docs.google.com/docs/3.9.1/general/options.html)
  + [Padding](http://docs.google.com/docs/3.9.1/general/padding.html)
  + [Performance](http://docs.google.com/docs/3.9.1/general/performance.html)
* Configuration
* Chart Types
* Axes
* Developers

[**#**](#gjdgxs) Performance

Chart.js charts are rendered on canvas elements, which makes rendering quite fast. For large datasets or performance sensitive applications, you may wish to consider the tips below.

## [**#**](#30j0zll) Data structure and format

### [**#**](#1fob9te) Parsing

Provide prepared data in the internal format accepted by the dataset and scales, and set parsing: false. See [Data structures](http://docs.google.com/docs/3.9.1/general/data-structures.html) for more information.

### [**#**](#3znysh7) Data normalization

Chart.js is fastest if you provide data with indices that are unique, sorted, and consistent across datasets and provide the normalized: true option to let Chart.js know that you have done so. Even without this option, it can sometimes still be faster to provide sorted data.

### [**#**](#2et92p0) Decimation

Decimating your data will achieve the best results. When there is a lot of data to display on the graph, it doesn't make sense to show tens of thousands of data points on a graph that is only a few hundred pixels wide.

The [decimation plugin](http://docs.google.com/docs/3.9.1/configuration/decimation.html) can be used with line charts to decimate data before the chart is rendered. This will provide the best performance since it will reduce the memory needed to render the chart.

Line charts are able to do [automatic data decimation during draw](#tyjcwt), when certain conditions are met. You should still consider decimating data yourself before passing it in for maximum performance since the automatic decimation occurs late in the chart life cycle.

## [**#**](#3dy6vkm) Tick Calculation

### [**#**](#1t3h5sf) Rotation

[Specify a rotation value](http://docs.google.com/docs/3.9.1/axes/cartesian/#tick-configuration) by setting minRotation and maxRotation to the same value, which avoids the chart from having to automatically determine a value to use.

### [**#**](#4d34og8) Sampling

Set the [ticks.sampleSize](http://docs.google.com/docs/3.9.1/axes/cartesian/#tick-configuration) option. This will determine how large your labels are by looking at only a subset of them in order to render axes more quickly. This works best if there is not a large variance in the size of your labels.

## [**#**](#2s8eyo1) Disable Animations

If your charts have long render times, it is a good idea to disable animations. Doing so will mean that the chart needs to only be rendered once during an update instead of multiple times. This will have the effect of reducing CPU usage and improving general page performance. Line charts use Path2D caching when animations are disabled and Path2D is available.

To disable animations

new Chart(ctx, {  
 type: 'line',  
 data: data,  
 options: {  
 animation: false  
 }  
});

## [**#**](#17dp8vu) Specify min and max for scales

If you specify the min and max, the scale does not have to compute the range from the data.

new Chart(ctx, {  
 type: 'line',  
 data: data,  
 options: {  
 scales: {  
 x: {  
 type: 'time',  
 min: new Date('2019-01-01').valueOf(),  
 max: new Date('2019-12-31').valueOf()  
 },  
 y: {  
 type: 'linear',  
 min: 0,  
 max: 100  
 }  
 }  
 }  
});

## [**#**](#3rdcrjn) Parallel rendering with web workers (Chromium only)

Chromium (Chrome: version 69, Edge: 79, Opera: 56) added the ability to [transfer rendering control of a canvas (opens new window)](https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/transferControlToOffscreen) to a web worker. Web workers can use the [OffscreenCanvas API (opens new window)](https://developer.mozilla.org/en-US/docs/Web/API/OffscreenCanvas) to render from a web worker onto canvases in the DOM. Chart.js is a canvas-based library and supports rendering in a web worker - just pass an OffscreenCanvas into the Chart constructor instead of a Canvas element. Note that as of today, this API is only supported in Chromium based browsers.

By moving all Chart.js calculations onto a separate thread, the main thread can be freed up for other uses. Some tips and tricks when using Chart.js in a web worker:

* Transferring data between threads can be expensive, so ensure that your config and data objects are as small as possible. Try generating them on the worker side if you can (workers can make HTTP requests!) or passing them to your worker as ArrayBuffers, which can be transferred quickly from one thread to another.
* You can't transfer functions between threads, so if your config object includes functions you'll have to strip them out before transferring and then add them back later.
* You can't access the DOM from worker threads, so Chart.js plugins that use the DOM (including any mouse interactions) will likely not work.
* Ensure that you have a fallback if you support browsers other than the most modern Chromium browsers.
* Resizing the chart must be done manually. See an example in the worker code below.

Example main thread code:

const config = {};  
const canvas = new HTMLCanvasElement();  
const offscreenCanvas = canvas.transferControlToOffscreen();  
const worker = new Worker('worker.js');  
worker.postMessage({canvas: offscreenCanvas, config}, [offscreenCanvas]);

Example worker code, in worker.js:

onmessage = function(event) {  
 const {canvas, config} = event.data;  
 const chart = new Chart(canvas, config);  
 // Resizing the chart must be done manually, since OffscreenCanvas does not include event listeners.  
 canvas.width = 100;  
 canvas.height = 100;  
 chart.resize();  
};

## [**#**](#26in1rg) Line Charts

### [**#**](#lnxbz9) Leave Bézier curves disabled

If you are drawing lines on your chart, disabling Bézier curves will improve render times since drawing a straight line is more performant than a Bézier curve. Bézier curves are disabled by default.

### [**#**](#tyjcwt) Automatic data decimation during draw

Line element will automatically decimate data, when tension, stepped, and borderDash are left set to their default values (false, 0, and [] respectively). This improves rendering speed by skipping drawing of invisible line segments.

### [**#**](#35nkun2) Enable spanGaps

If you have a lot of data points, it can be more performant to enable spanGaps. This disables segmentation of the line, which can be an unneeded step.

To enable spanGaps:

new Chart(ctx, {  
 type: 'line',  
 data: {  
 datasets: [{  
 spanGaps: true // enable for a single dataset  
 }]  
 },  
 options: {  
 spanGaps: true // enable for all datasets  
 }  
});

### [**#**](#1ksv4uv) Disable Line Drawing

If you have a lot of data points, it can be more performant to disable rendering of the line for a dataset and only draw points. Doing this means that there is less to draw on the canvas which will improve render performance.

To disable lines:

new Chart(ctx, {  
 type: 'line',  
 data: {  
 datasets: [{  
 showLine: false // disable for a single dataset  
 }]  
 },  
 options: {  
 showLine: false // disable for all datasets  
 }  
});

### [**#**](#44sinio) Disable Point Drawing

If you have a lot of data points, it can be more performant to disable rendering of the points for a dataset and only draw line. Doing this means that there is less to draw on the canvas which will improve render performance.

To disable point drawing:

new Chart(ctx, {  
 type: 'line',  
 data: {  
 datasets: [{  
 pointRadius: 0 // disable for a single dataset  
 }]  
 },  
 options: {  
 datasets: {  
 line: {  
 pointRadius: 0 // disable for all `'line'` datasets  
 }  
 },  
 elements: {  
 point: {  
 radius: 0 // default to disabled in all datasets  
 }  
 }  
 }  
});

## [**#**](#2jxsxqh) When transpiling with Babel, consider using loose mode

Babel 7.9 changed the way classes are constructed. It is slow, unless used with loose mode. [More information (opens new window)](https://github.com/babel/babel/issues/11356)

Last Updated: 8/3/2022, 12:46:38 PM

←  [Padding](http://docs.google.com/docs/3.9.1/general/padding.html)   [Configuration](http://docs.google.com/docs/3.9.1/configuration/)  →