Stochastic Gradient Descent Part I

Linear Regression

Previously on CSCI 4622

Given data $(x_{i1}, x_{i2}, \ldots, x_{ip}, y_i)$ for $i = 1, 2, \ldots, n$ fit a MLR model of the form

$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \cdots + \beta_p x_{ip} + \epsilon_i \qquad \text{where} \qquad \epsilon_i \sim N(0, \sigma^2)$$

by minimizing $$\mathrm{RSS}_\lambda = \sum_{i=1}^{n} \left[(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) - y_i\right]^2 + \lambda \sum_{k=1}^{p} \beta_k^2$$

Given data $(x_{i1}, x_{i2}, \ldots, x_{ip}, y_i)$ for $i = 1, 2, \ldots, n$ fit a LogReg model of the form

$$p(y = 1 \mid x) = \mathrm{sigm}(\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p)$$

as we'll see later this week, by minimizing a similar loss function.

Finding Parameters in Linear Regression

Whether doing simple linear regression or multiple linear regression, parameters are estimated by minimizing the RSS loss function.

$$\mathrm{RSS}_\lambda = \sum_{i=1}^{n} [(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) - y_i]^2 + \lambda \sum_{k=1}^{p} \beta_k^2$$

When you lots of data and a model with many features, this becomes a difficult problem

While direct methods (based on linear algebra) exist, they are far too memory and computationally expensive to perform in real life

Instead, we use an **iterative method**

Iterative Solution Methods

Iterative methods can be thought of as very intelligent guess and check

$$y_i = \beta_0 + \beta_1 x_i$$

- Make a guess at the parameters
- Update your guess in a smart way, based on the problem specs, to get a better guess
- Repeat until guess converges to something very close to the correct answer

$$\beta_0^{(0)} \to \beta_0^{(1)} \to \cdots \to \beta_0^{(k)} \approx \beta_0$$

$$\beta_1^{(0)} \to \beta_1^{(1)} \to \cdots \to \beta_1^{(k)} \approx \beta_1$$

A Silly Example

Suppose you want to find the minimum of the function $f(z) = z^2 - 2z + 2$

We can rewrite in a slightly better form:

minimizer is $z=1$

$$f(z) = (z-1)^2 + 1$$

Question: What nice properties for minimization does this function have?

A Silly Example

Suppose you want to find the minimum of the function $f(z) = z^2 - 2z + 2$

OK, suppose that I guess that the minimizer is $z^{(0)} = 2.25$

Question: Which way should I move?

$f'(z) = 2z - 2$

$f'(2.25) =$

$2 \cdot 2.25 - 2$

$= 2.5$

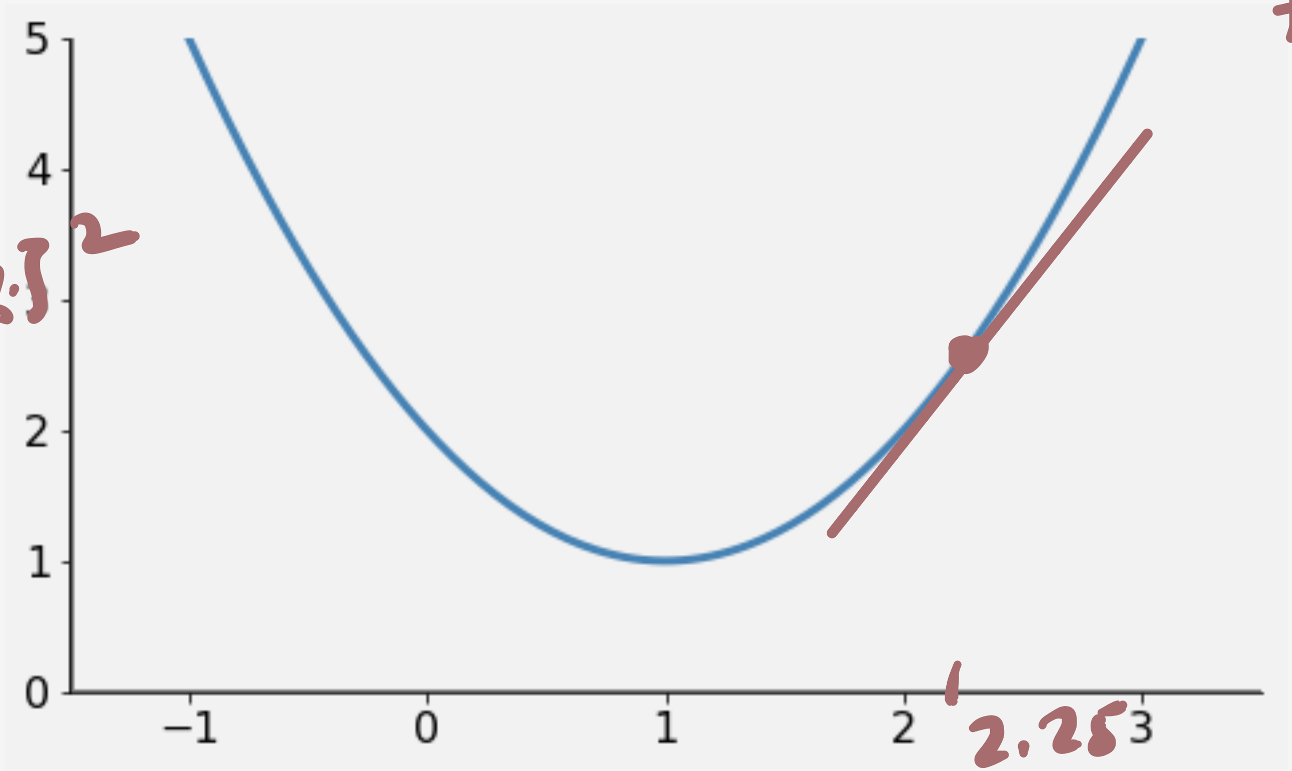
f' is > 0

move LEFT

$f' < 0$

$\rightarrow$ move RIGHT

RSS $= \sum_{i=1}^{n} [(\beta_0 + \beta_1 x_i) - y_i]^2$

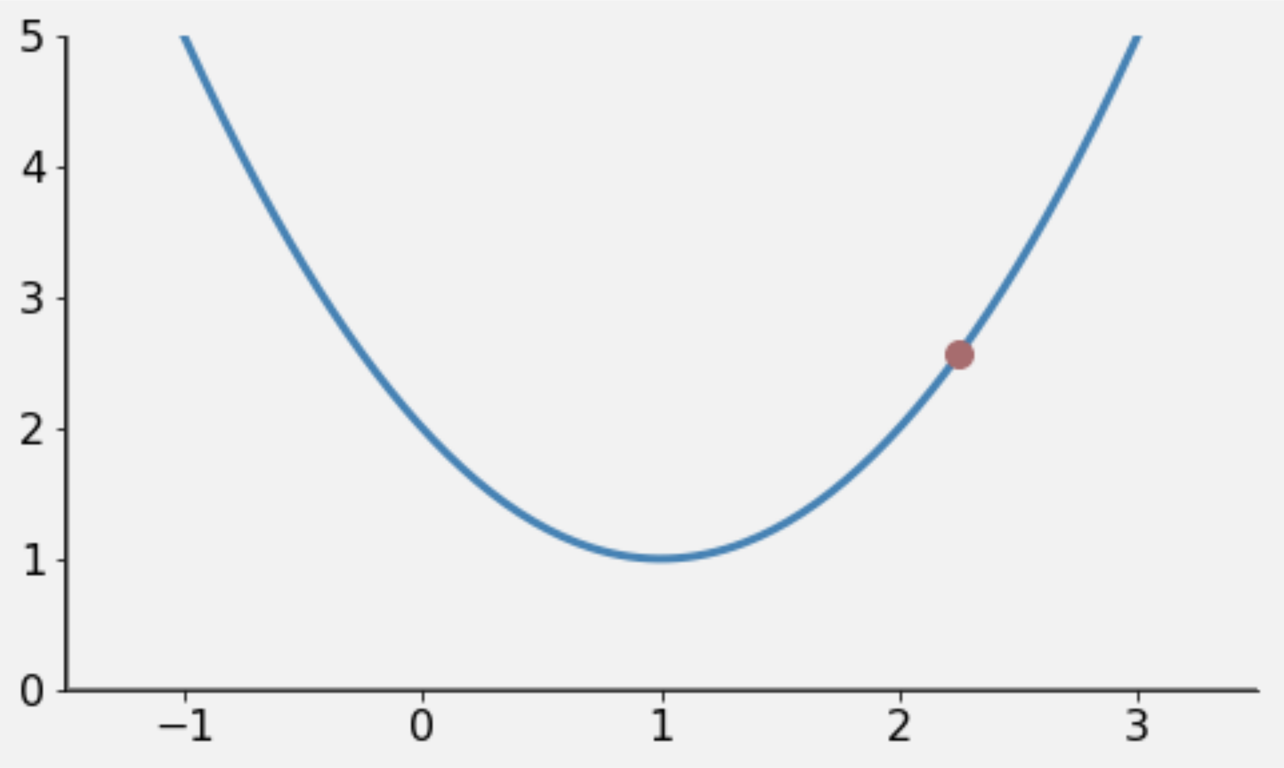


A Silly Example

Suppose you want to find the minimum of the function $f(z) = z^2 - 2z + 2$

OK, suppose that I guess that the minimizer is $z^{(0)} = 2.25$

Question: Which way should I move? **Answer**: Downhill! But which way is down?



A Silly Example

Suppose you want to find the minimum of the function $f(z) = z^2 - 2z + 2$

OK, suppose that I guess that the minimizer is $z^{(0)} = 2.25$

Question: But which way is down? **Answer**: Derivative tells you uphill. Go opposite direction

A Silly Example

Suppose you want to find the minimum of the function $f(z) = z^2 - 2z + 2$

Question: But which way is down? **Answer**: Derivative tells you uphill. Go opposite direction

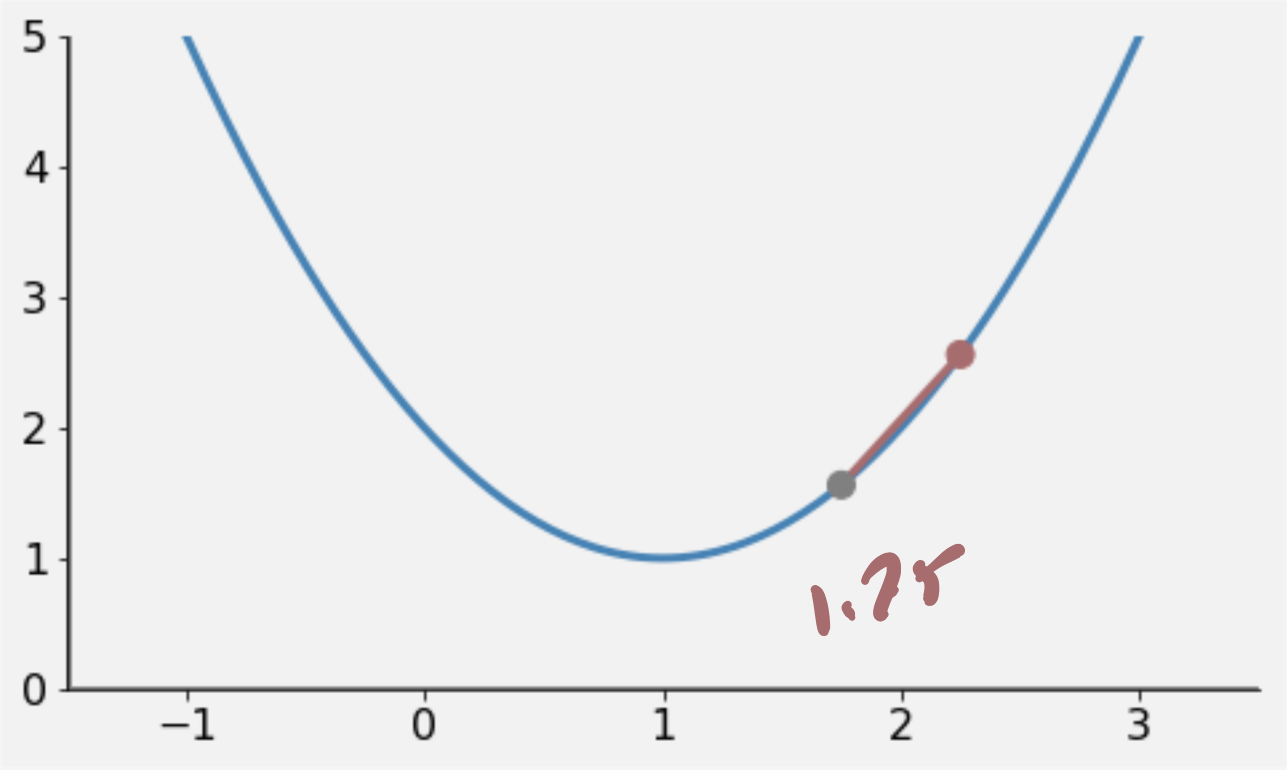
Question: How far should we move? **Answer**: Dunno, just pick a small step size like $\eta = 0.5$

A Silly Example

Suppose you want to find the minimum of the function $f(z) = z^2 - 2z + 2$

Question: But which way is down? **Answer**: Derivative tells you uphill. Go opposite direction

Question: How far should we move? **Answer**: Dunno, just pick a small step size like $\eta = 0.5$

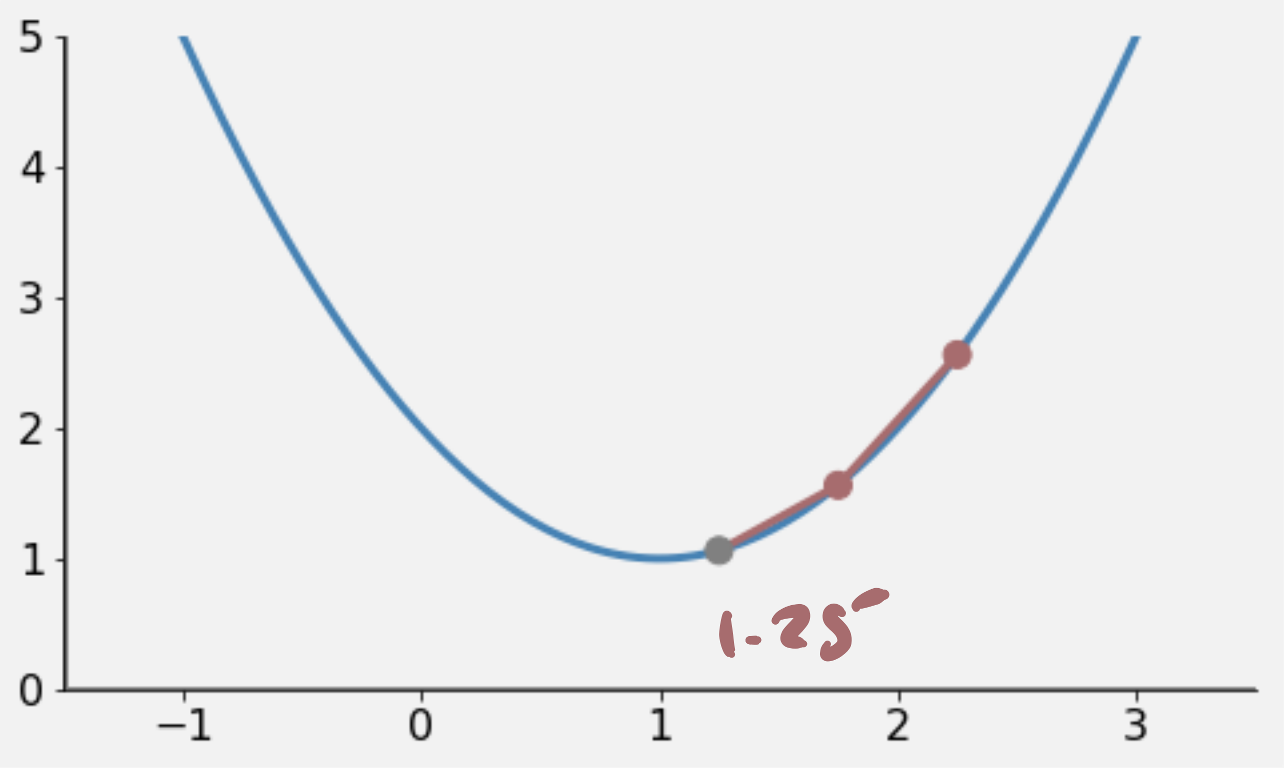


A Silly Example

Suppose you want to find the minimum of the function $f(z) = z^2 - 2z + 2$

Question: But which way is down? **Answer**: Derivative tells you uphill. Go opposite direction

Question: How far should we move? **Answer**: Dunno, just pick a small step size like $\eta = 0.5$

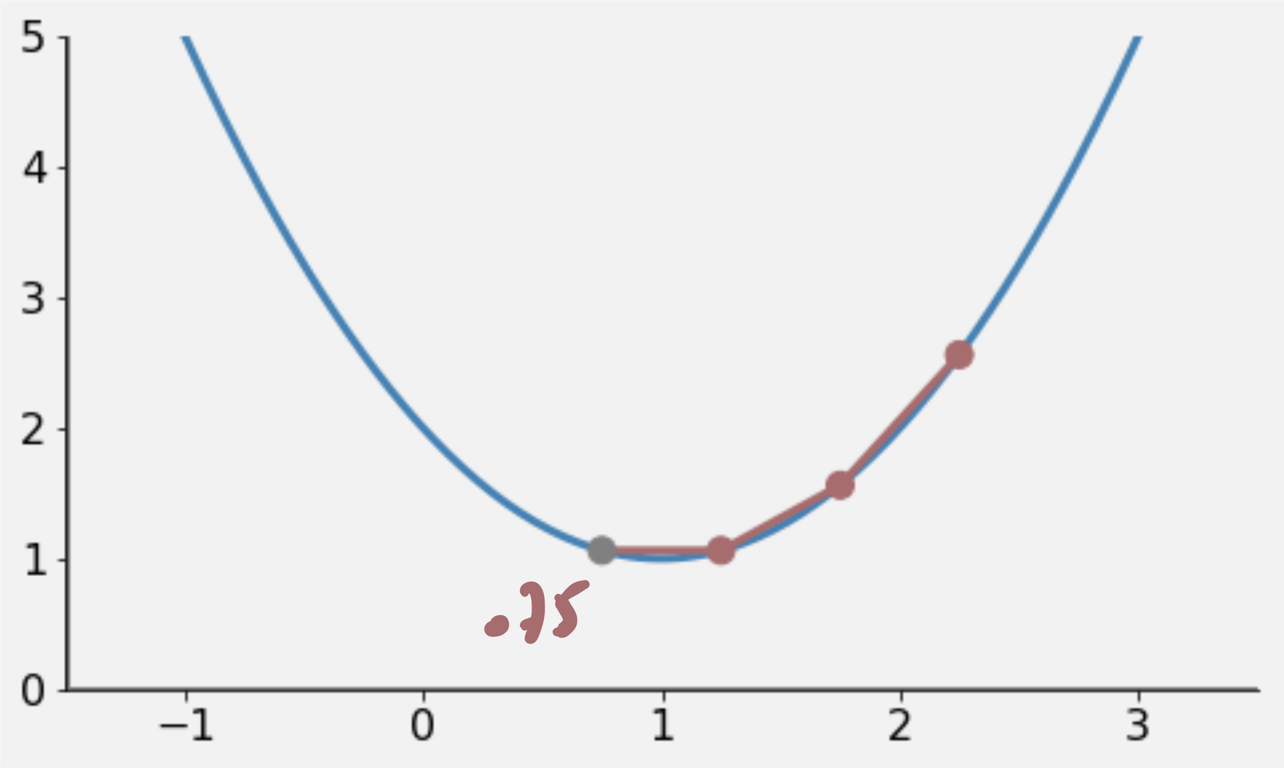


A Silly Example

Suppose you want to find the minimum of the function $f(z) = z^2 - 2z + 2$

Question: But which way is down? **Answer**: Derivative tells you uphill. Go opposite direction

Question: How far should we move? **Answer**: Dunno, just pick a small step size like $\eta = 0.5$

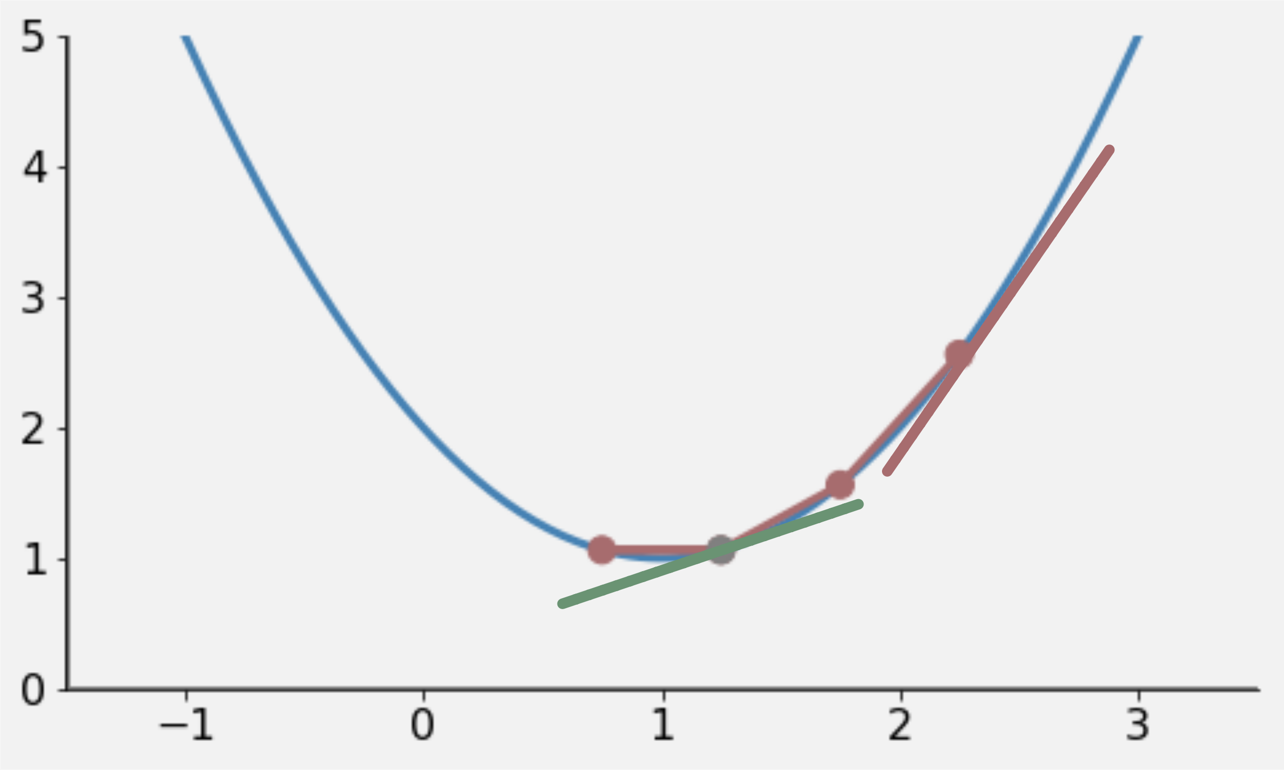


A Silly Example

Suppose you want to find the minimum of the function $f(z) = z^2 - 2z + 2$

Question: But which way is down? **Answer**: Derivative tells you uphill. Go opposite direction

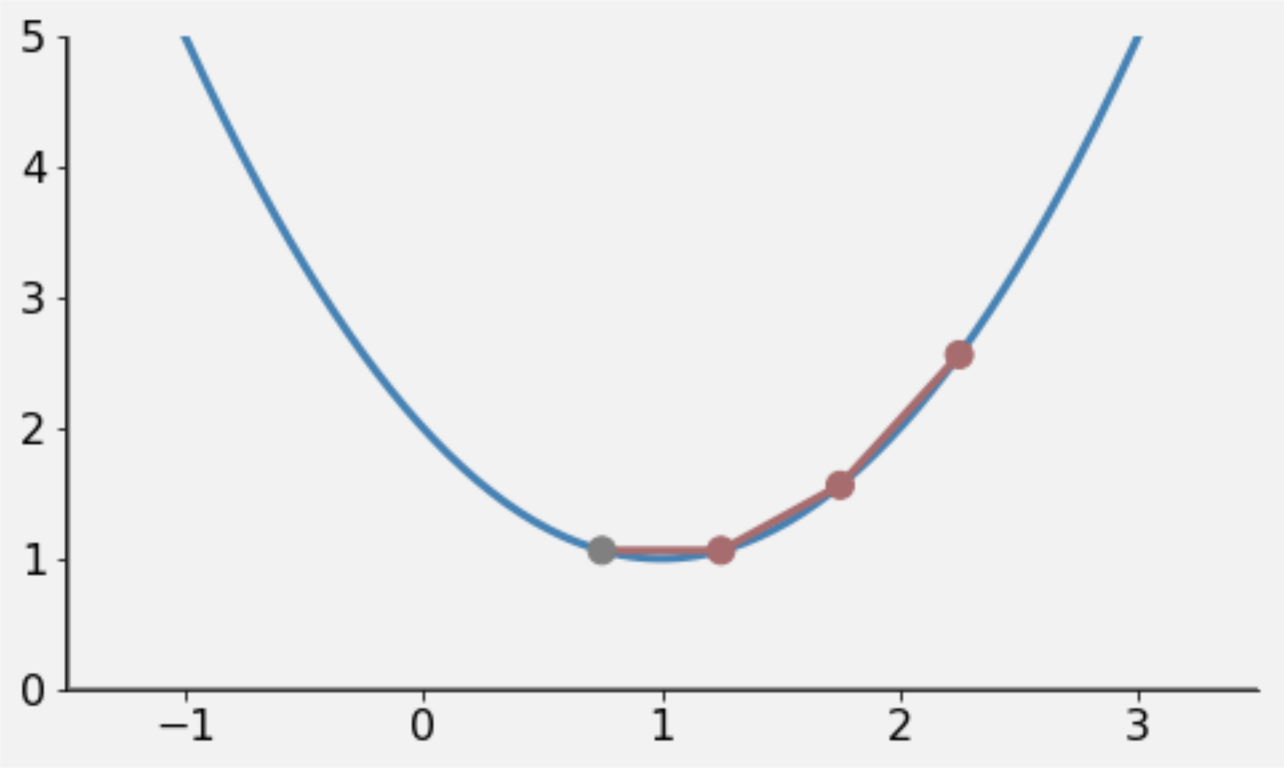
Question: How far should we move? **Answer**: Dunno, just pick a small step size like $\eta = 0.5$



A Silly Example

Question: How do we fix this so we can get closer to the true minimum?

- BAD IDEA: REDUCE LEARNING RATE
- GOOD IDEA: MAKE STEP PROP to DERIV. SIZE

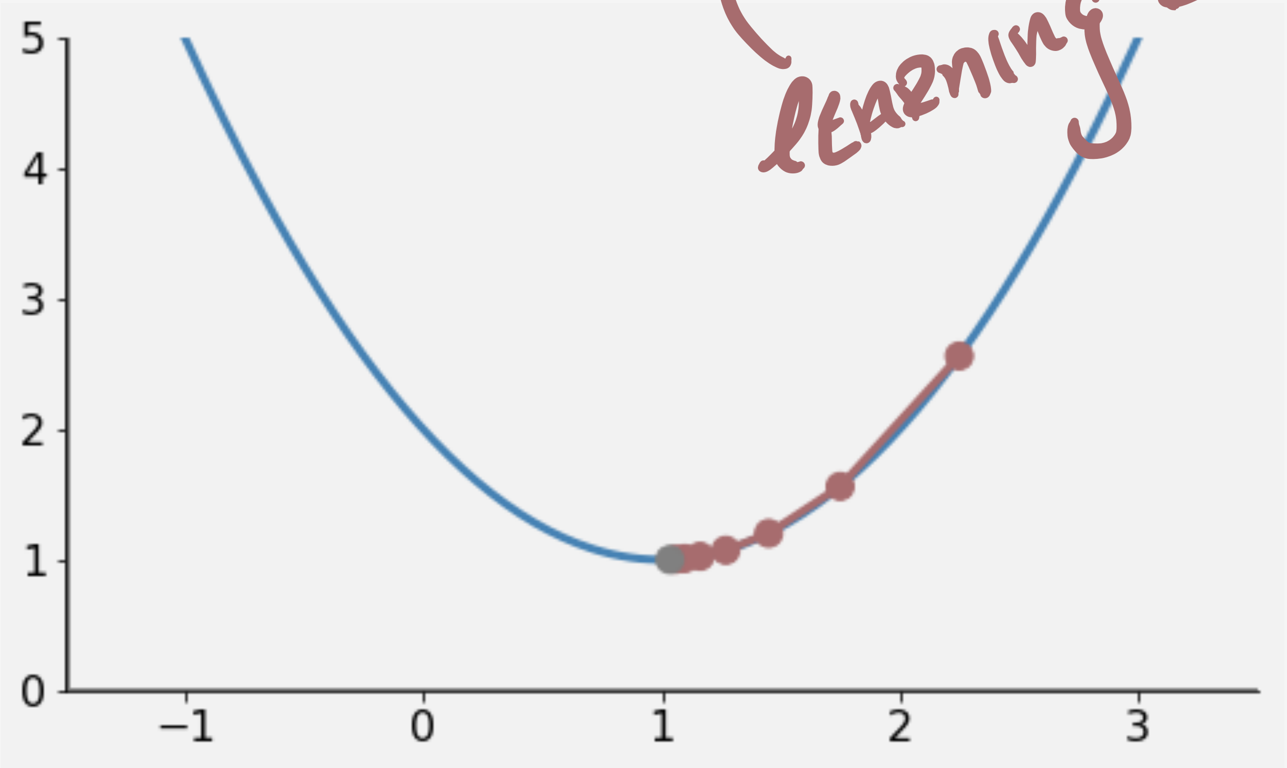


A Silly Example

This method is called **Gradient Descent** (think Derivative Descent)

$$z^{(k+1)} = z^{(k)} - \eta \cdot f'(z^{(k)})$$

LEARNING RATE



Simple Linear Regression

OK, so how do we use the idea of Gradient Descent to estimate the parameters in SLR

Recall, the estimated parameters are the value of β_0 and β_1 that minimize the RSS

$$\text{RSS} = \sum_{i=1}^{n} \left[(\beta_0 + \beta_1 x_i) - y_i\right]^2$$

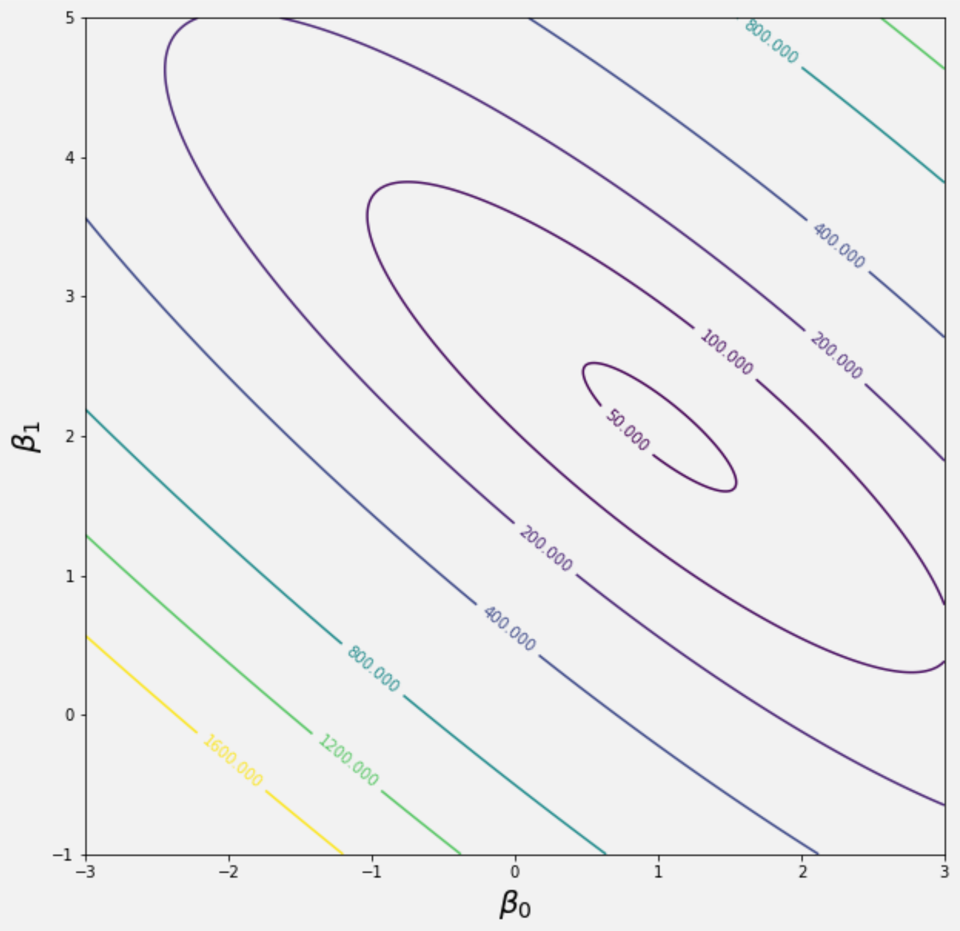
Important: We're minimizing over the β's . The x's and y's are the values from the data.

The difficulty is that this is a function of two variables, which is not quite a simple parabola

Simple Linear Regression

In 1-dimension the RSS is a parabola. In 2-dimensions it's a bowl-like surface

$$\mathrm{RSS} = \sum_{i=1}^{n} \left[(\beta_0 + \beta_1 x_i) - y_i\right]^2$$



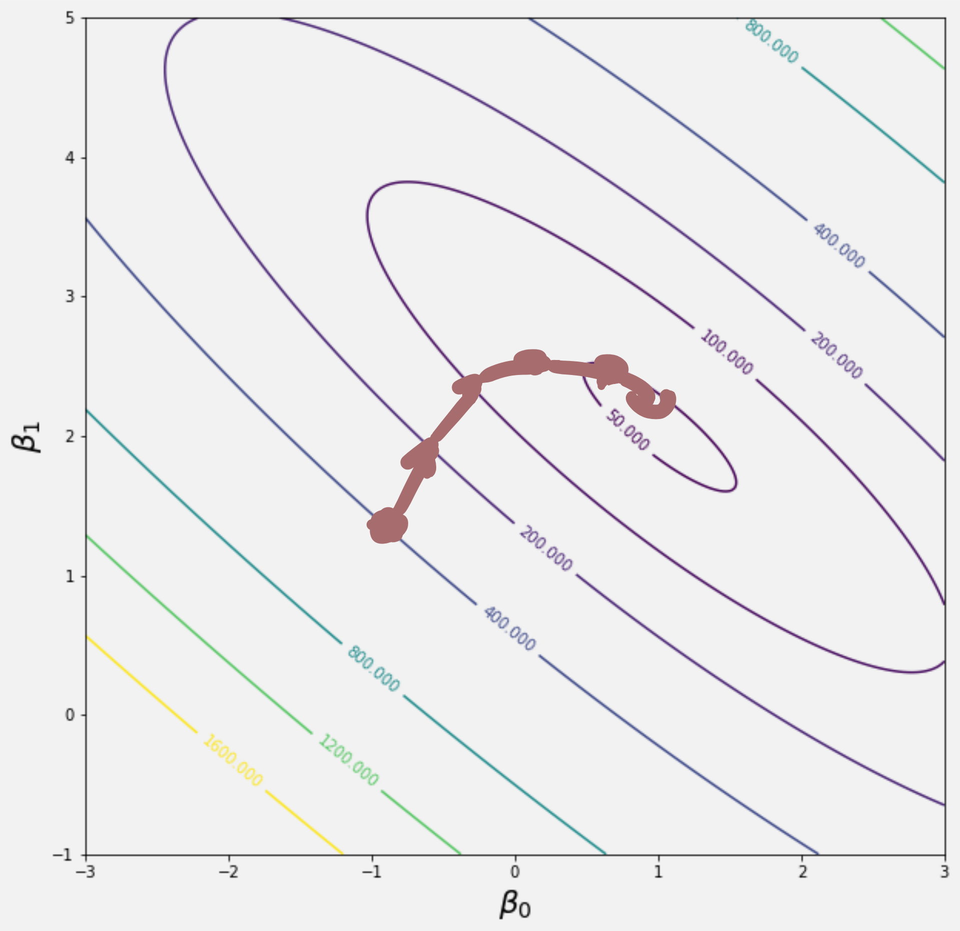
Simple Linear Regression

In 1-dimension the RSS is a parabola. In 2-dimensions it's a bowl-like surface

$$\mathrm{RSS} = \sum_{i=1}^{n} \left[(\beta_0 + \beta_1 x_i) - y_i\right]^2$$

In 2D the process is still the same:

- Make a guess at the minimum
- Move iteratively downhill



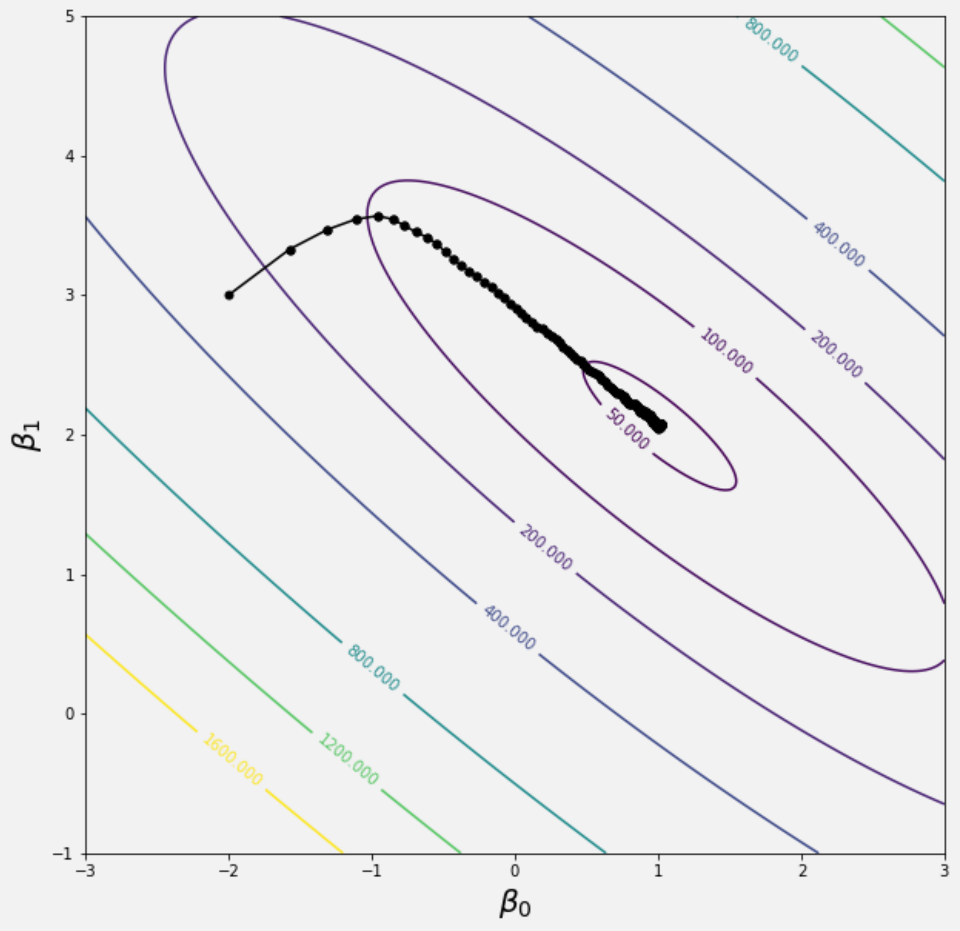
Simple Linear Regression

In 1-dimension the RSS is a parabola. In 2-dimensions it's a bowl-like surface

$$\text{RSS} = \sum_{i=1}^{n} \left[(\beta_0 + \beta_1 x_i) - y_i\right]^2$$

In 2D the process is still the same:

- Make a guess at the minimum
- Move iteratively downhill

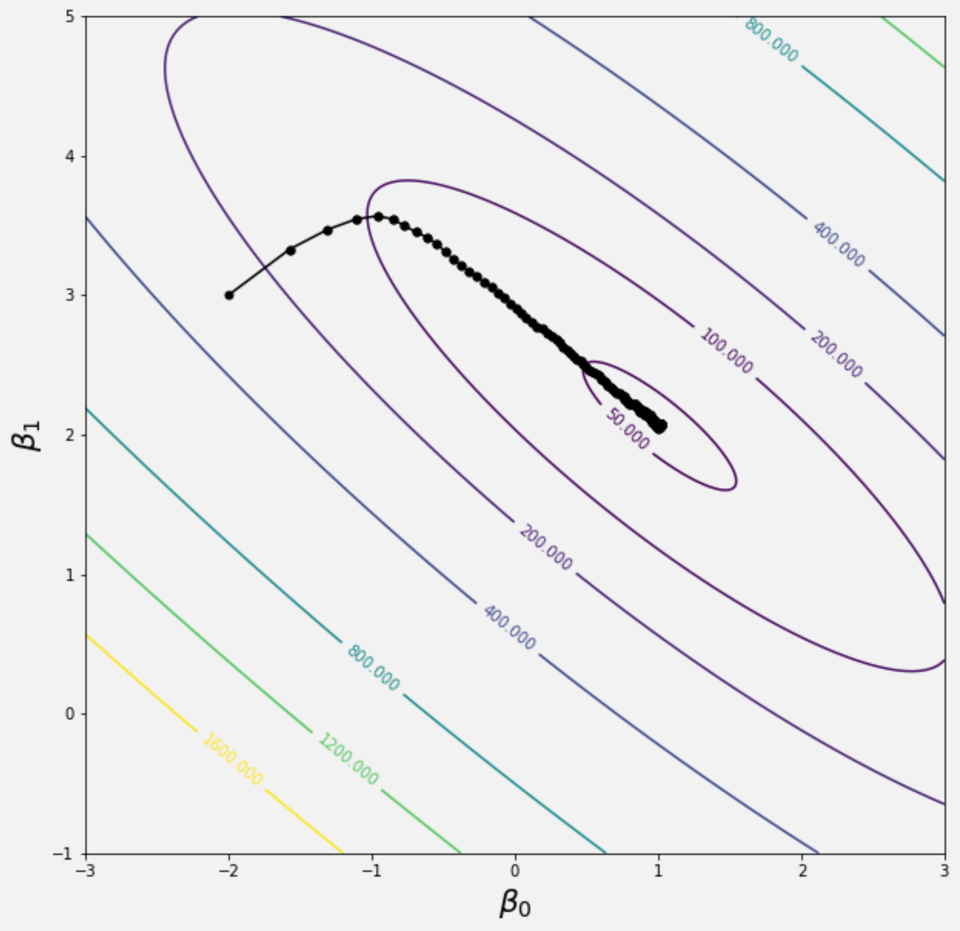


Simple Linear Regression

In 1-dimension the RSS is a parabola. In 2-dimensions it's a bowl-like surface

$$\text{RSS} = \sum_{i=1}^{n} \left[(\beta_0 + \beta_1 x_i) - y_i\right]^2$$

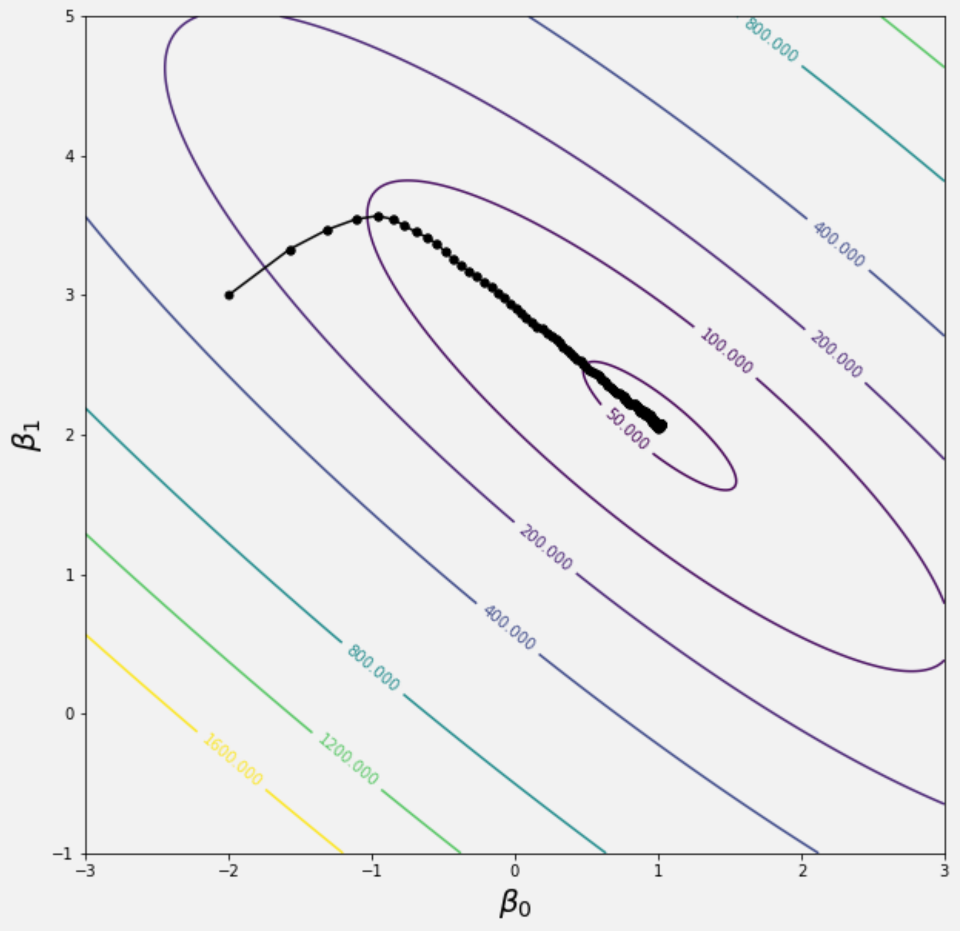
But how do we move downhill in 2d?



Simple Linear Regression

In 1-dimension the RSS is a parabola. In 2-dimensions it's a bowl-like surface

$$\text{RSS} = \sum_{i=1}^{n} \left[(\beta_0 + \beta_1 x_i) - y_i \right]^2$$



But how do we move downhill in 2d?

- Move downhill in each coordinate direction
- Use partial derivatives

Gradient Descent for SLR

In 1-dimension, Gradient Descent was $z^{(k+1)} = z^{(k)} - \eta \cdot f'(z^{(k)})$

In 2-dimensions, we have the following:

$$\beta_0 \quad \leftarrow \quad \beta_0 - \eta \frac{\partial \mathrm{RSS}}{\partial \beta_0}$$

$$\beta_1 \quad \leftarrow \quad \beta_1 - \eta \frac{\partial \mathrm{RSS}}{\partial \beta_1}$$

$$RSS = \sum_{i=1}^{n} [(\beta_0 + \beta_1 x_i) - y_i]^2$$

$$\frac{\partial RSS}{\partial \beta_0} = \sum_{i=1}^{n} 2[(\beta_0 + \beta_1 x_i) - y_i] \cdot 1$$

$$\frac{\partial RSS}{\partial \beta_1} = \sum_{i=1}^{n} 2[(\beta_0 + \beta_1 x_i) - y_i] x_i$$

Gradient Descent for SLR

In 1-dimension, Gradient Descent was $z^{(k+1)} = z^{(k)} - \eta \cdot f'(z^{(k)})$

In 2-dimensions, we have the following:

$$\beta_0 \quad \leftarrow \quad \beta_0 - \eta \sum_{i=1}^{n} 2\left[(\beta_0 + \beta_1 x_i) - y_i\right]$$

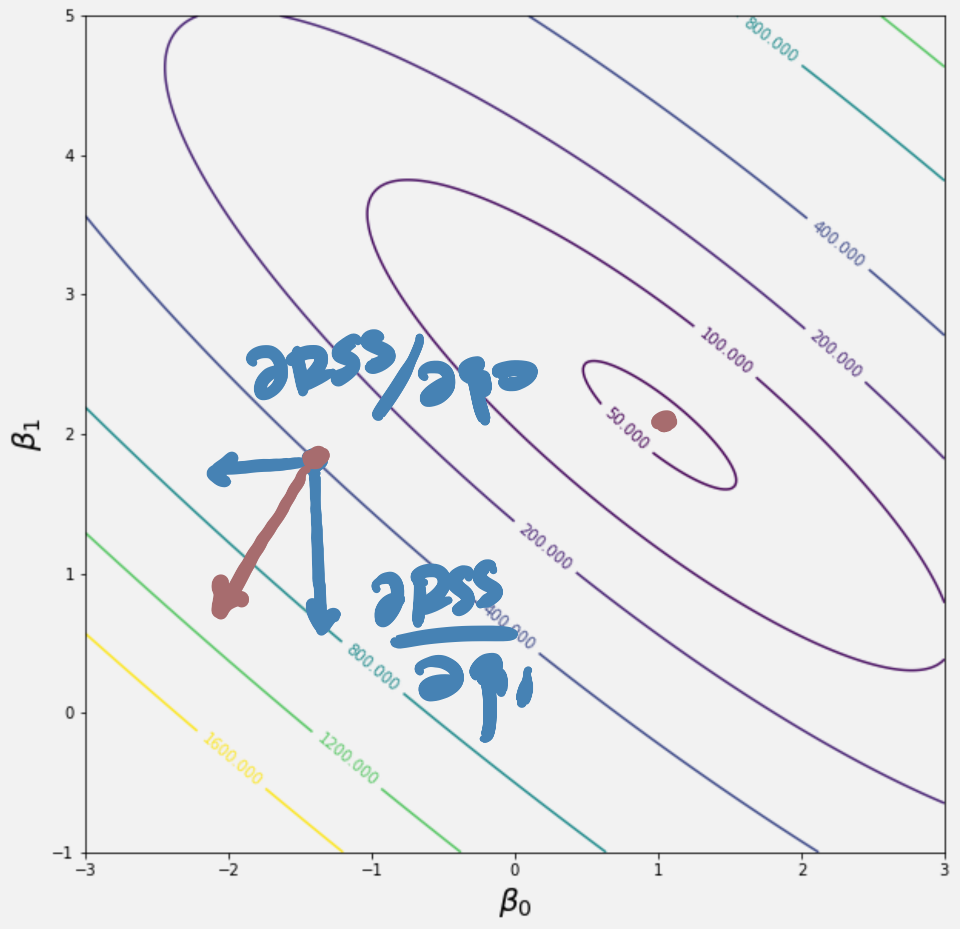
$$\beta_1 \quad \leftarrow \quad \beta_1 - \eta \sum_{i=1}^{n} 2\left[(\beta_0 + \beta_1 x_i) - y_i\right] x_i$$

Vectorizing Gradient Descent

Let $\boldsymbol{\beta} = [\beta_0, \beta_1]^T$ be vector of the parameters. Need to **vectorize** derivatives too

$$\frac{\partial \text{RSS}}{\partial \beta_0} = \sum_{i=1}^{n} 2\left[(\beta_0 + \beta_1 x_i) - y_i\right]$$

$$\frac{\partial \text{RSS}}{\partial \beta_1} = \sum_{i=1}^{n} 2\left[(\beta_0 + \beta_1 x_i) - y_i\right] x_i$$



Vector of derivatives is called the **Gradient**

Points in direction that loss function increases the **fastest** (i.e. steepest direction)

Vectorizing Gradient Descent

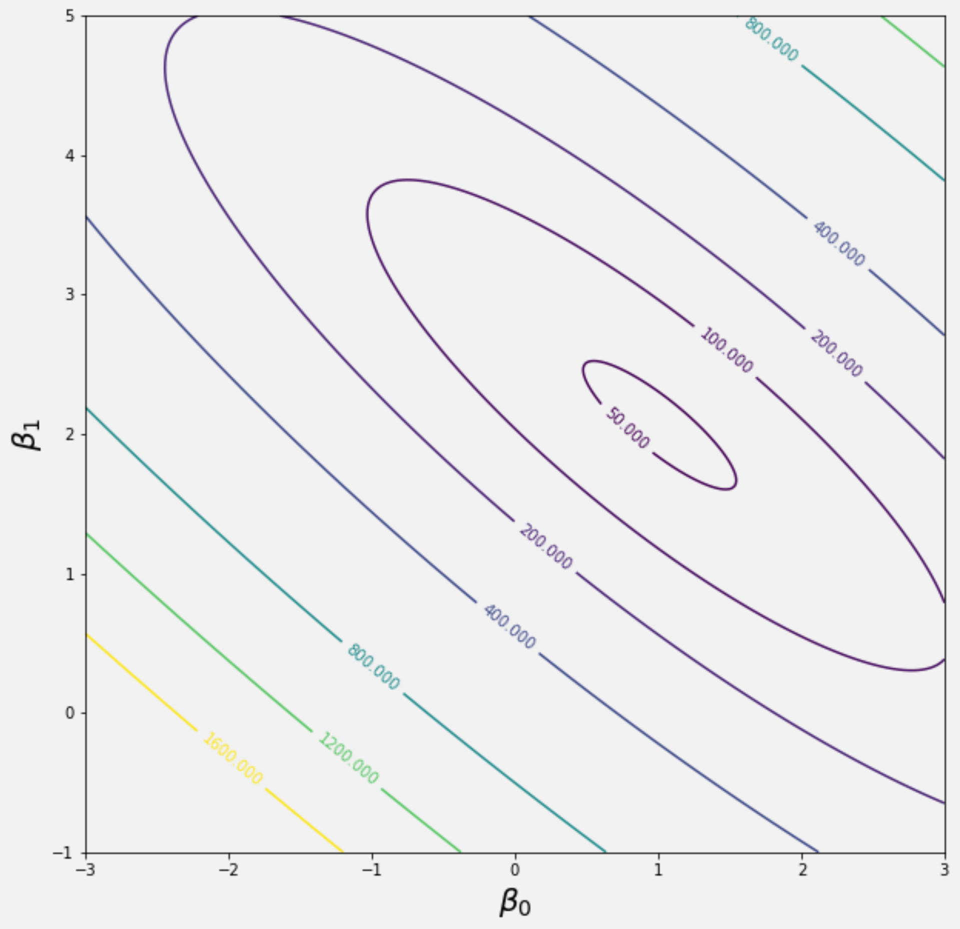
Let $\boldsymbol{\beta} = [\beta_0, \beta_1]^T$ be vector of the parameters. Need to **vectorize** derivatives too

$$\nabla \mathrm{RSS} = \begin{bmatrix} \sum_{i=1}^{n} 2\left[(\beta_0 + \beta_1 x_i) - y_i\right] \\ \sum_{i=1}^{n} 2\left[(\beta_0 + \beta_1 x_i) - y_i\right] x_i \end{bmatrix}$$

$$\boxed{\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta \nabla \mathrm{RSS}}$$

Vector of derivatives is called the **Gradient**

Points in direction that loss function increases the **fastest** (i.e. steepest direction)



Vectorized Gradient Descent for SLR

OK, so we guess values of the parameters, and then perform gradient updates to improve

$$\begin{bmatrix} \beta_0 \\ \\ \beta_1 \end{bmatrix} \leftarrow \begin{bmatrix} \beta_0 \\ \\ \beta_1 \end{bmatrix} - \eta \begin{bmatrix} \sum_{i=1}^n 2\left[(\beta_0 + \beta_1 x_i) - y_i\right] \\ \\ \sum_{i=1}^n 2\left[(\beta_0 + \beta_1 x_i) - y_i\right] x_i \end{bmatrix}$$

Question: Does anything about this seem slow?

Stochastic Gradient Descent

Update based on one training example at a time. Faster, but less accurate updates

FOR ONE TRAINING EX (x_i, y_i)

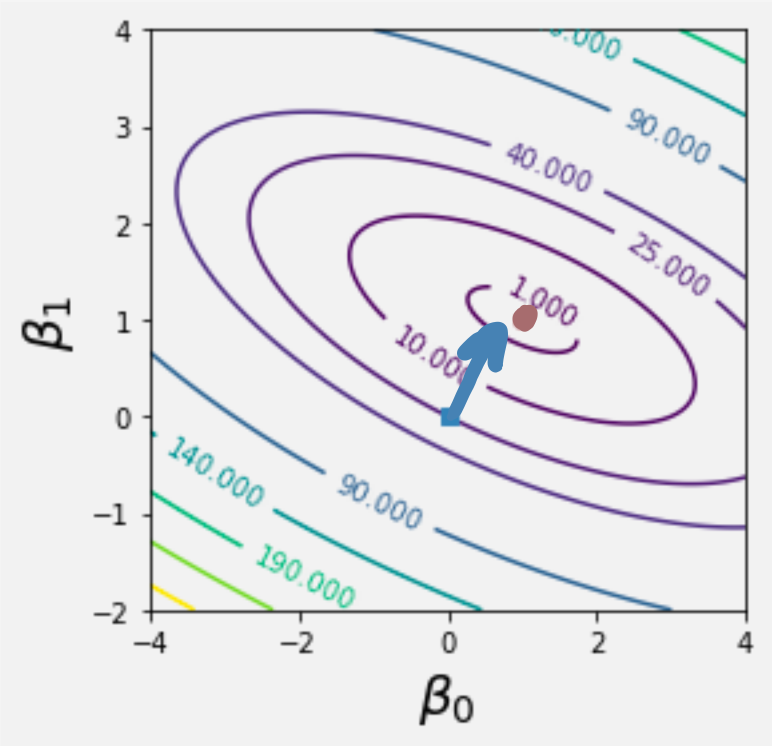
$$\begin{bmatrix} \beta_0 \\ \\ \beta_1 \end{bmatrix} \leftarrow \begin{bmatrix} \beta_0 \\ \\ \beta_1 \end{bmatrix} - \eta \begin{bmatrix} 2\left((\beta_0 + \beta_1 x_i) - y_i\right) \\ \\ 2\left((\beta_0 + \beta_1 x_i) - y_i\right) x_i \end{bmatrix}$$

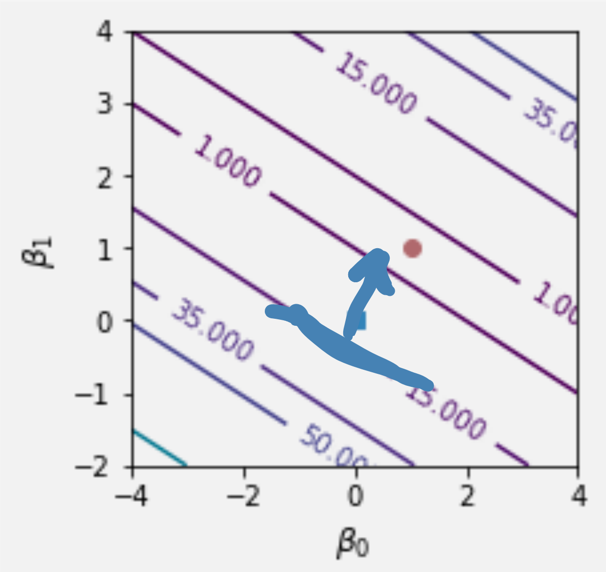
Important: Just like when training a Perceptron:

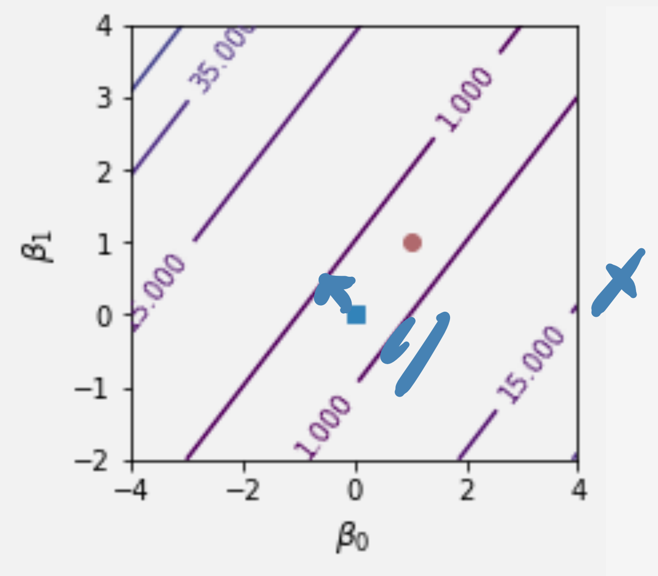
- Loop through training examples in random order to avoid bias
- Go through all training examples before starting over
- One pass through training examples is called an **Epoch**

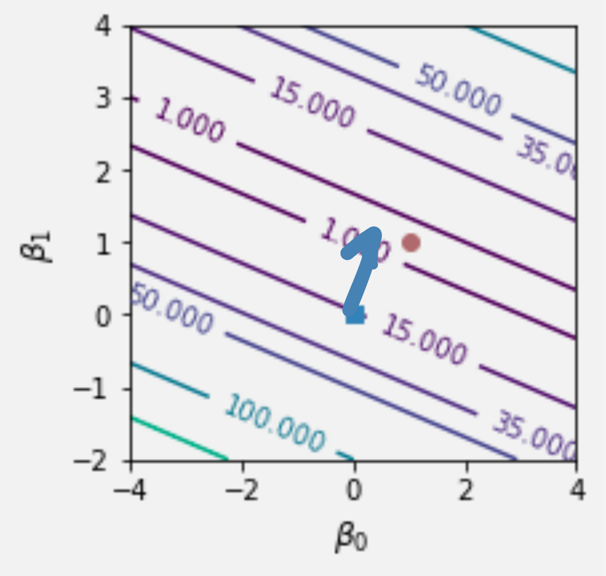
Geometry of Stochastic Gradient Descent

$$RSS_i = [(\beta_0 + \beta_1 x_i) - y_i]^2$$



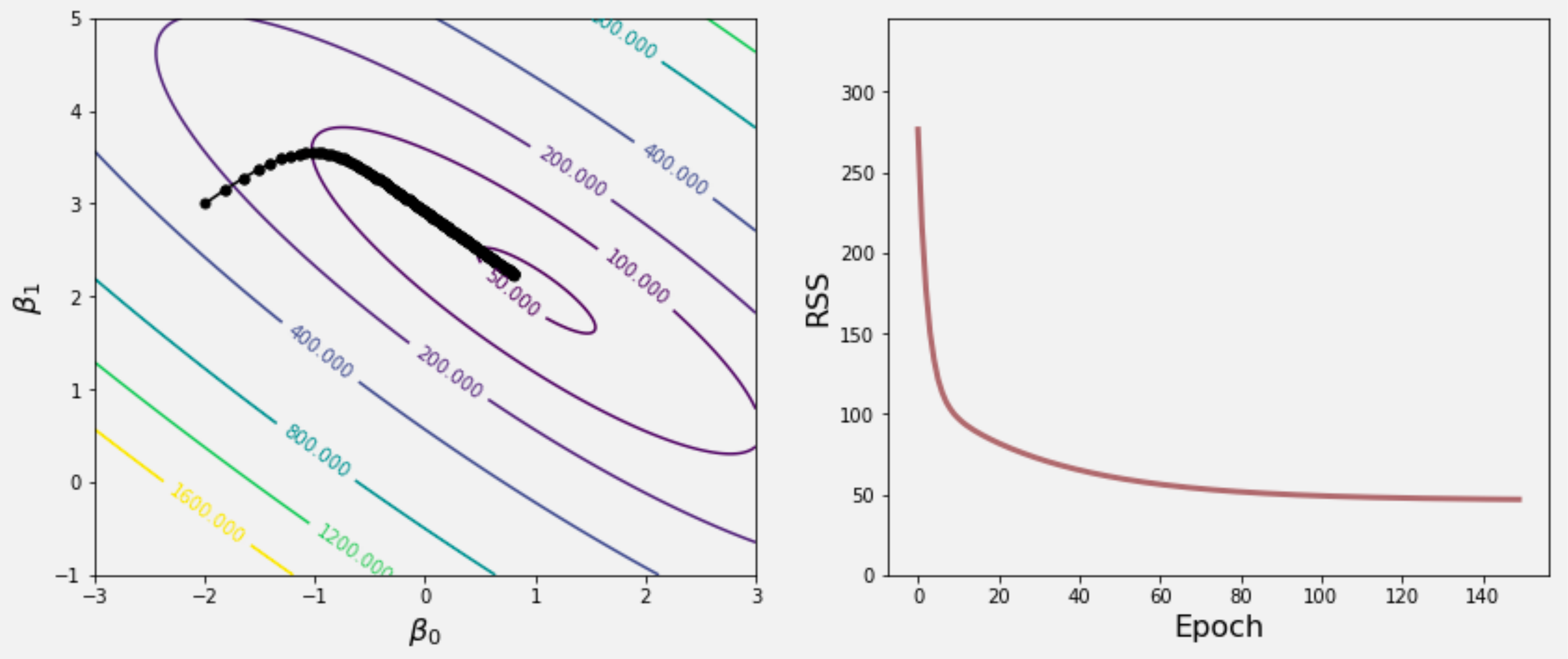






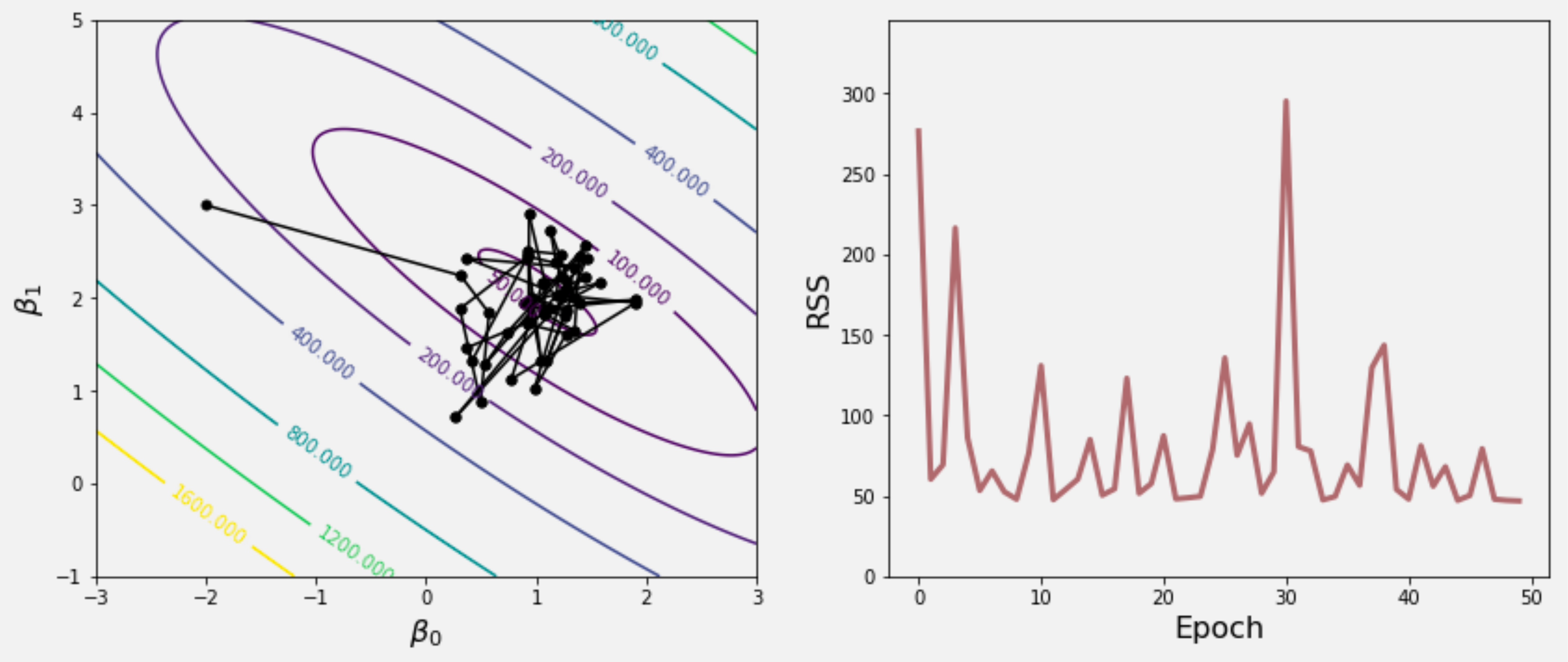
The Importance of the Learning Rate

Too small of a learning rate and it takes forever to converge



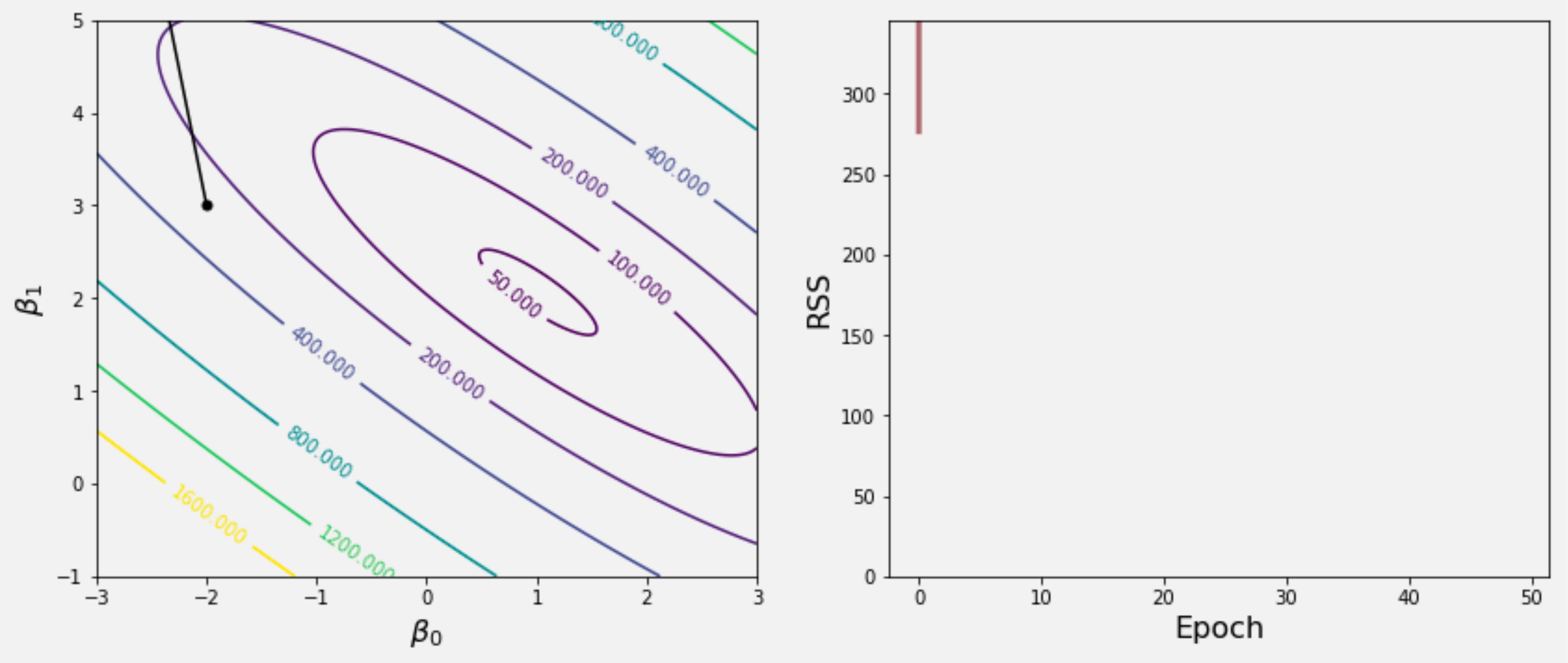
The Importance of the Learning Rate

Too large a learning rate and you can get oscillations that bounce around



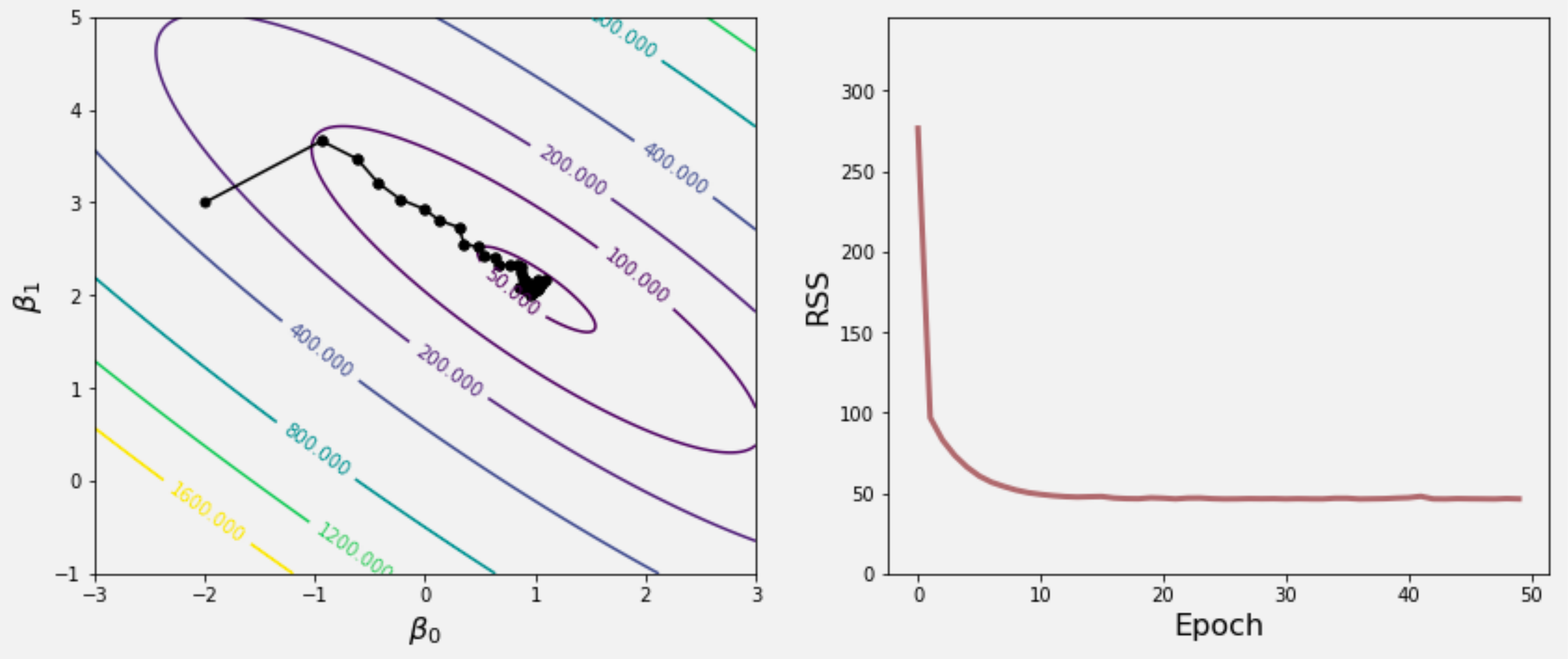
The Importance of the Learning Rate

Way too large a learning rate and you can diverge



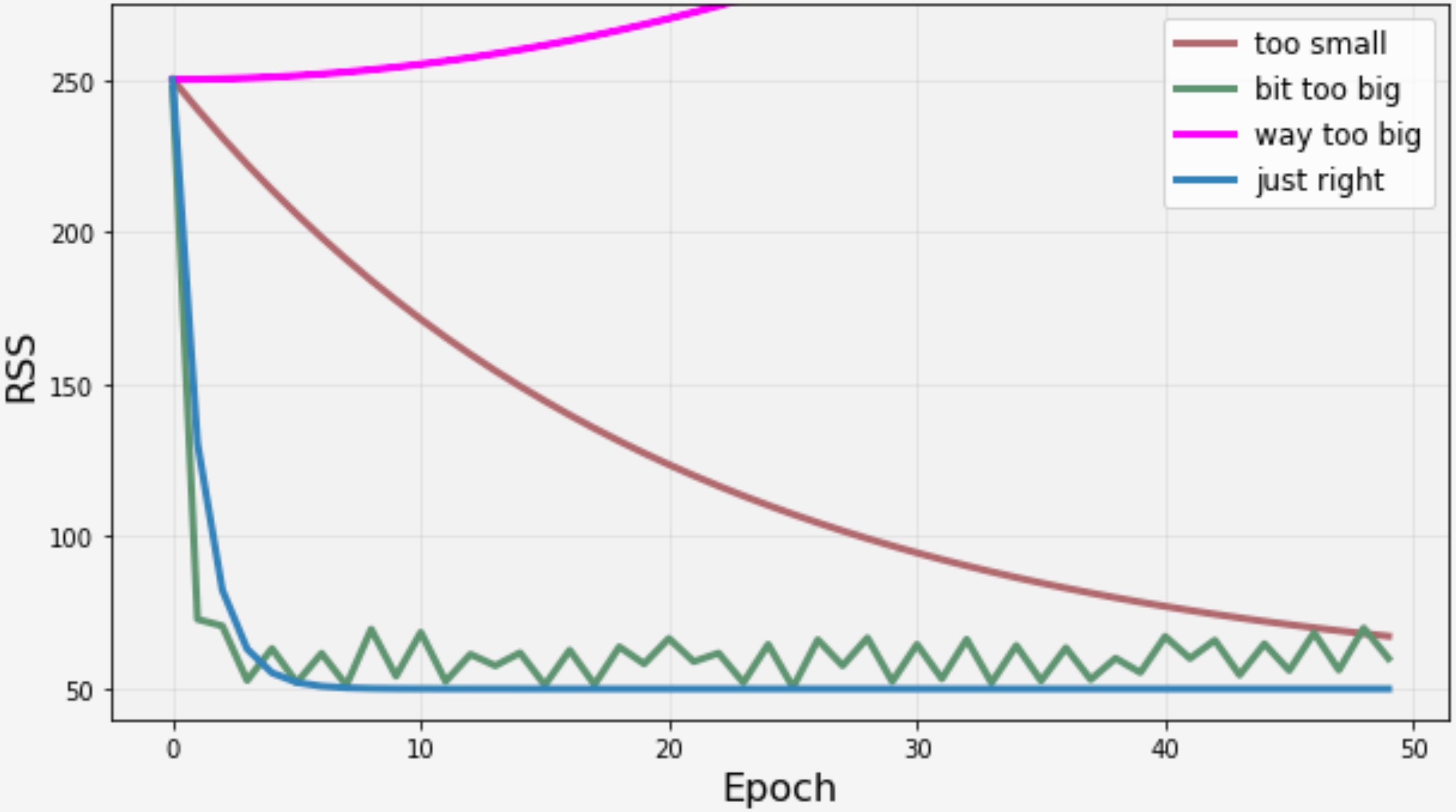
The Importance of the Learning Rate

Generally have to tune learning rate to get it just right



The Importance of the Learning Rate

Generally have to tune learning rate to get it just right



SGD in Higher Dimensions

So far we've looked at SGD for SLR, where we have two parameters

But what about in the more common case of multiple linear regression with p parameters?

$$\text{RSS} = \sum_{i=1}^{n} \left[(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) - y_i\right]^2$$

Still has same general form, but now the parameter vector and gradient vector are longer

$$\boldsymbol{\beta} \leftarrow \boldsymbol{\beta} - \eta \nabla \text{RSS} \quad \Leftrightarrow \quad \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} \leftarrow \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} - \eta \begin{bmatrix} 2((\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) - y_i) \\ 2((\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) - y_i) x_{i1} \\ \vdots \\ 2((\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) - y_i) x_{ip} \end{bmatrix}$$

SGD in Higher Dimensions

What if we want to add regularization? (PSST: We **always** want to add regularization)

$$\mathrm{RSS}_\lambda = \sum_{i=1}^{n} \left[(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) - y_i\right]^2 + \lambda \sum_{k=1}^{p} \beta_k^2$$

Only thing that changes is the derivatives of the non-bias parameters

$$\begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} \leftarrow \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} - \eta \begin{bmatrix} 2((\beta_0 + \beta_1 x_{i1} + \cdots + \beta_{\mathrm{p}} x_{ip}) - y_i) \\ 2((\beta_0 + \beta_1 x_{i1} + \cdots + \beta_{\mathrm{p}} x_{ip}) - y_i) x_{i1} + 2\lambda\beta_1 \\ \vdots \\ 2((\beta_0 + \beta_1 x_{i1} + \cdots + \beta_{\mathrm{p}} x_{ip}) - y_i) x_{ip} + 2\lambda\beta_p \end{bmatrix}$$

SGD Part I Wrap-Up

- SGD and it's variants are most popular way of training most learning models
- Fairly efficient in time. **Very** efficient in space.
- Lots of little improvements we can do to make more efficient (Hands-On Friday)

Next Time:

- See how we can do SGD for learning the weights in Logistic Regression
- Derive the Logistic Regression loss function via Maximum Likelihood Estimation

If-Time Bonus: Better Vectorization

Recall the Regularized SGD iteration for multiple linear regression

$$\mathrm{RSS}_\lambda = \sum_{i=1}^{n} \left[(\beta_0 + \beta_1 x_{i1} + \cdots + \beta_p x_{ip}) - y_i \right]^2 + \lambda \sum_{k=1}^{p} \beta_k^2$$

Do you see anything redundant in each term in the gradient?

$$\begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} \leftarrow \begin{bmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{bmatrix} - \eta \begin{bmatrix} 2((\beta_0 + \beta_1 x_{i1} + \cdots + \beta_1 x_{ip}) - y_i) \\ 2((\beta_0 + \beta_1 x_{i1} + \cdots + \beta_1 x_{ip}) - y_i) x_{i1} + 2\lambda\beta_1 \\ \vdots \\ 2((\beta_0 + \beta_1 x_{i1} + \cdots + \beta_1 x_{ip}) - y_i) x_{ip} + 2\lambda\beta_p \end{bmatrix}$$