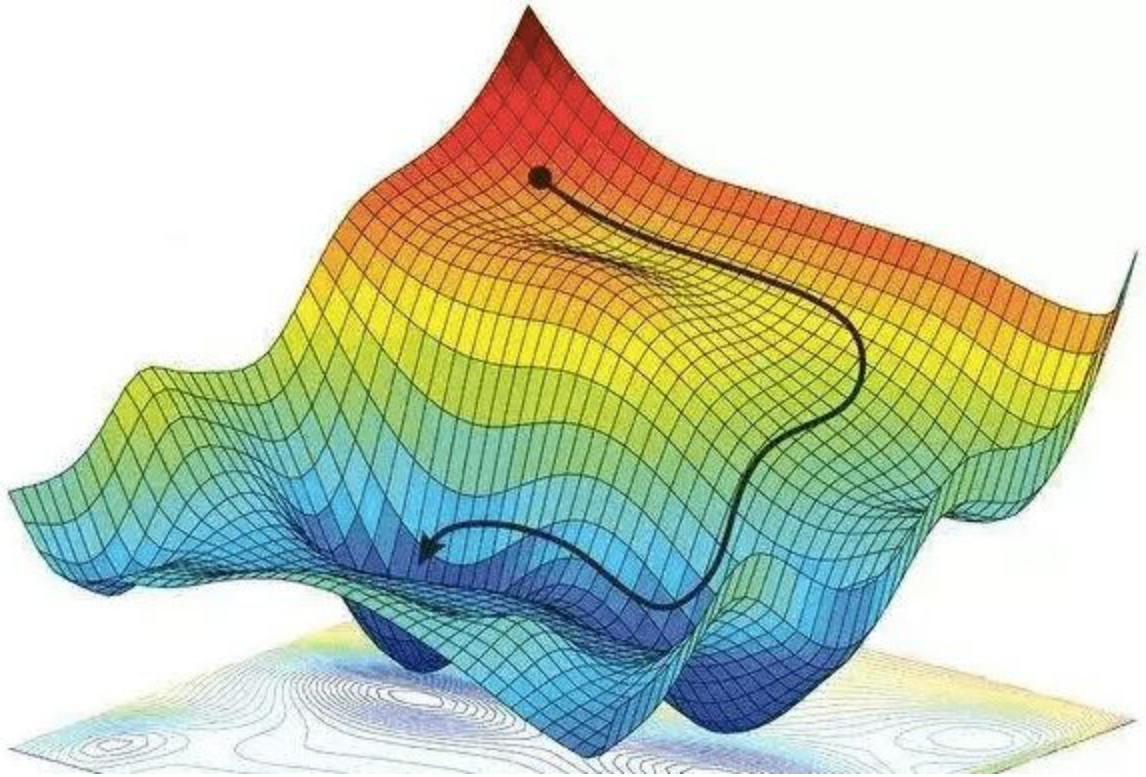
# Neural Networks and Optimization II

Dr. Parlett-Pelleriti

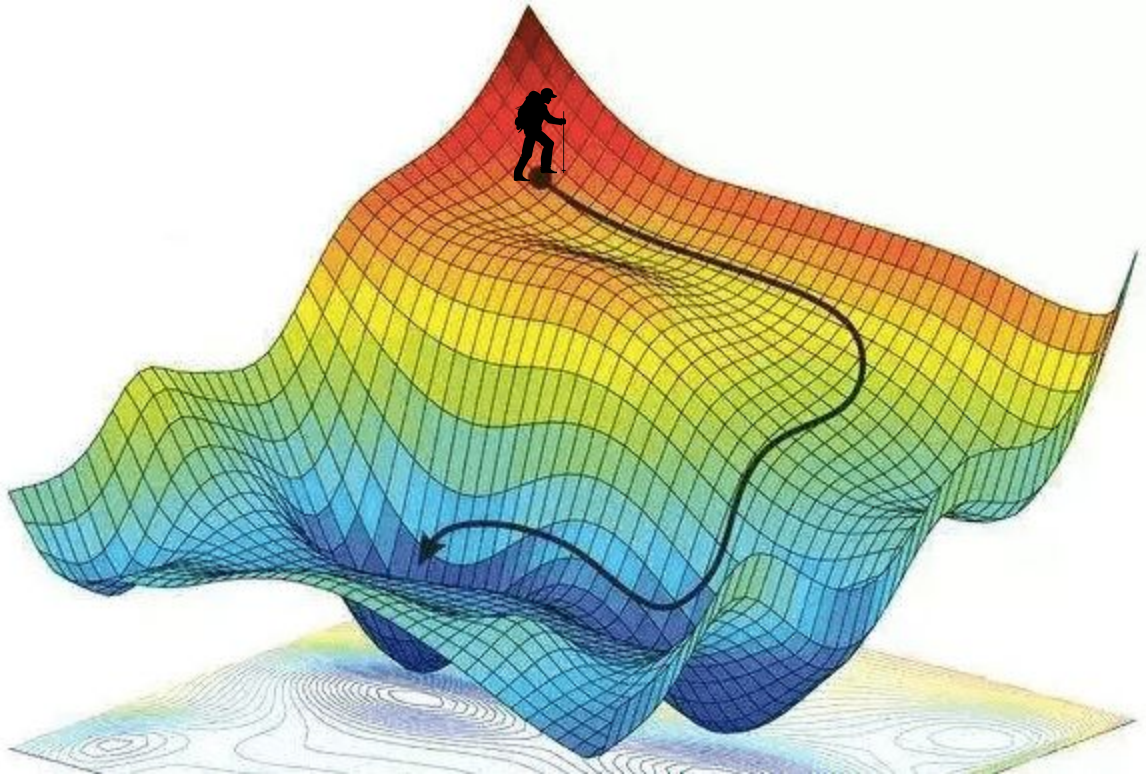
# Optimization

- Gradient Descent
- Stochastic Gradient Descent
- Momentum
- RMSP
- Adam

# Gradient Descent



# Gradient Descent



# Gradient Descent Ideas

1. The goal is to minimize the loss function

# Partial Derivatives

$$f = x^2 + xy + y^2$$

How does **f(x,y)** change when **x** changes? What about when **y** changes?

# Partial Derivatives

$$f = x^2 + xy + y^2$$

How does **f(x,y)** change when **x** changes? What about when **y** changes?

$$\frac{\partial f}{\partial x} = 2x + y \qquad \frac{\partial f}{\partial y} = x + 2y$$

# Gradient

$$f = x^2 + xy + y^2$$

How does **f(x,y)** change when **x** changes? What about when **y** changes?

$$\frac{\partial f}{\partial x} = 2x + y \qquad \frac{\partial f}{\partial y} = x + 2y$$

$$\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + y \\ x + 2y \end{bmatrix}$$

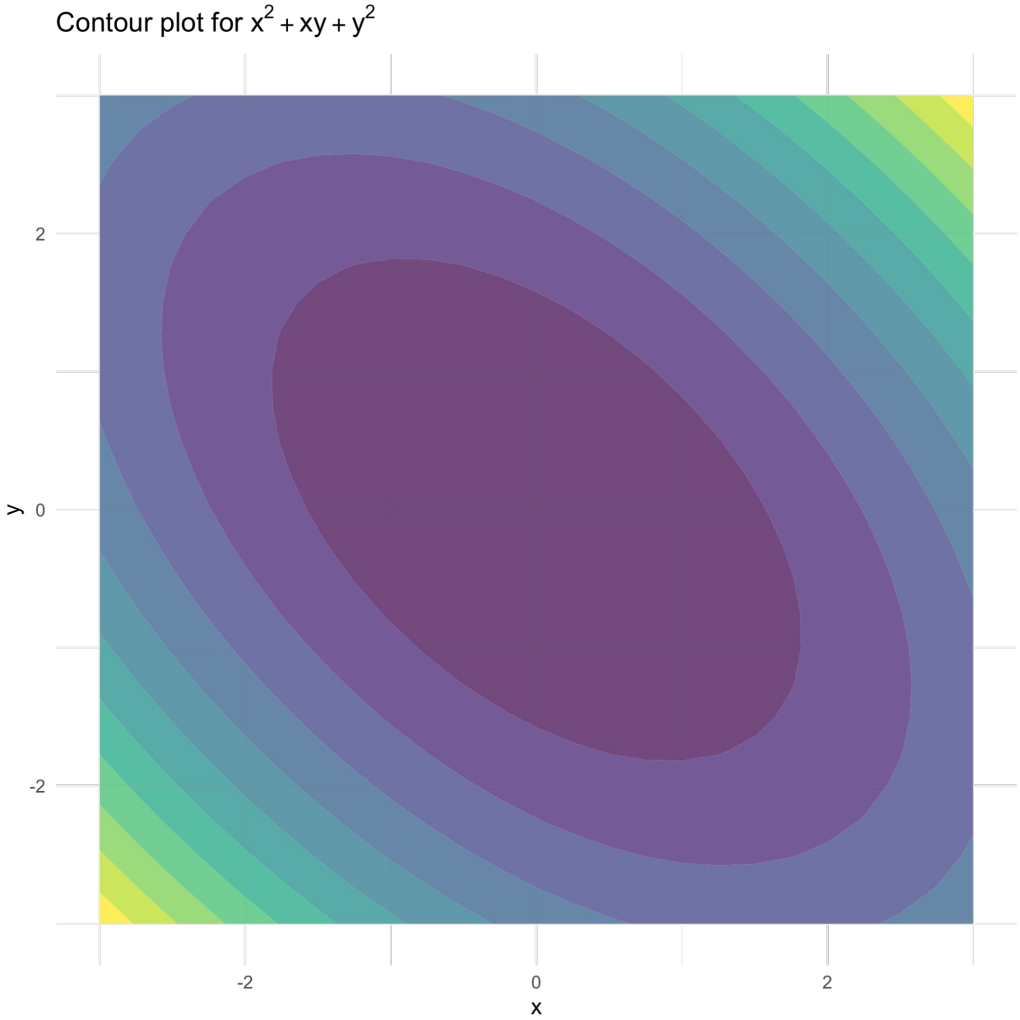
# Gradient

$$f = x^2 + xy + y^2$$

How does **f(x,y)** change when **x** changes? What about when **y** changes?

$$\frac{\partial f}{\partial x} = 2x + y \qquad \frac{\partial f}{\partial y} = x + 2y$$

$$\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + y \\ x + 2y \end{bmatrix}$$



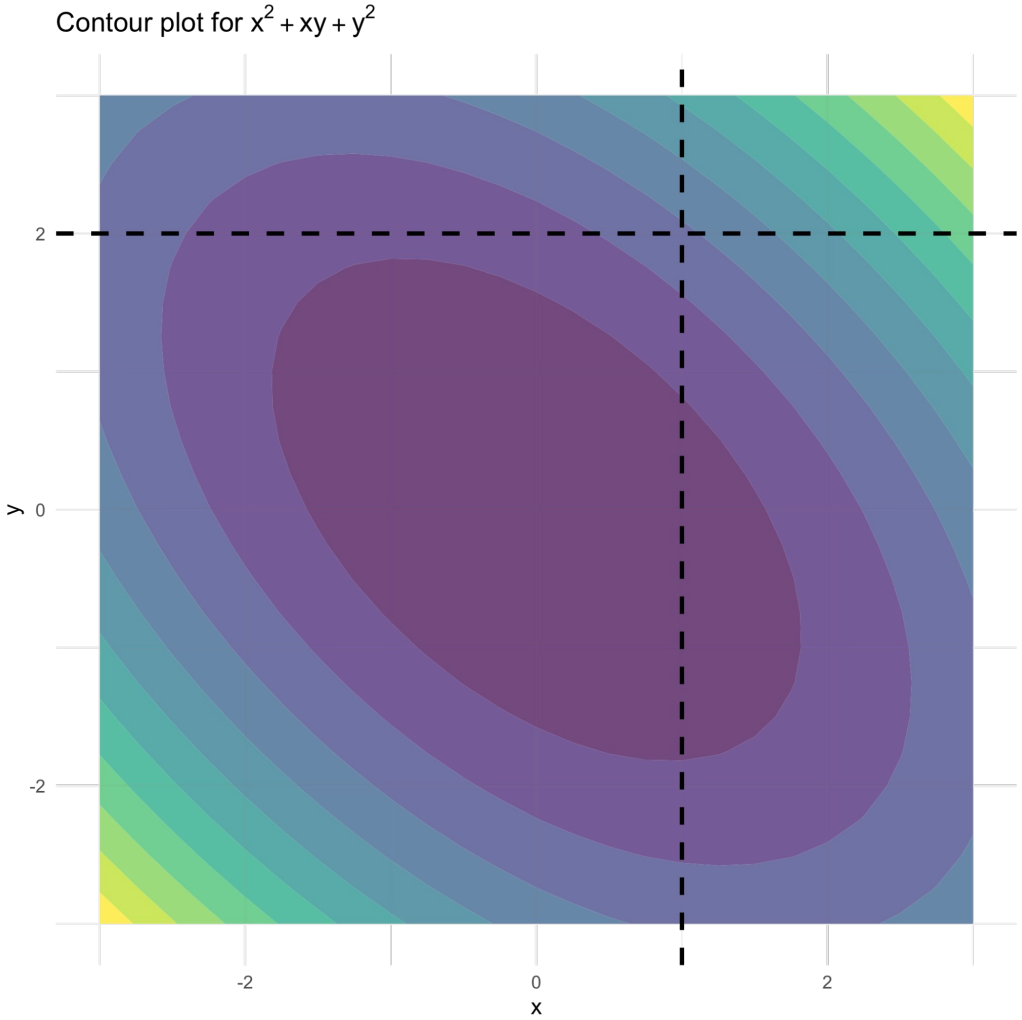
# Gradient

$$f = x^2 + xy + y^2$$

How does **f(x,y)** change when **x** changes? What about when **y** changes?

$$\frac{\partial f}{\partial x} = 2x + y \qquad \frac{\partial f}{\partial y} = x + 2y$$

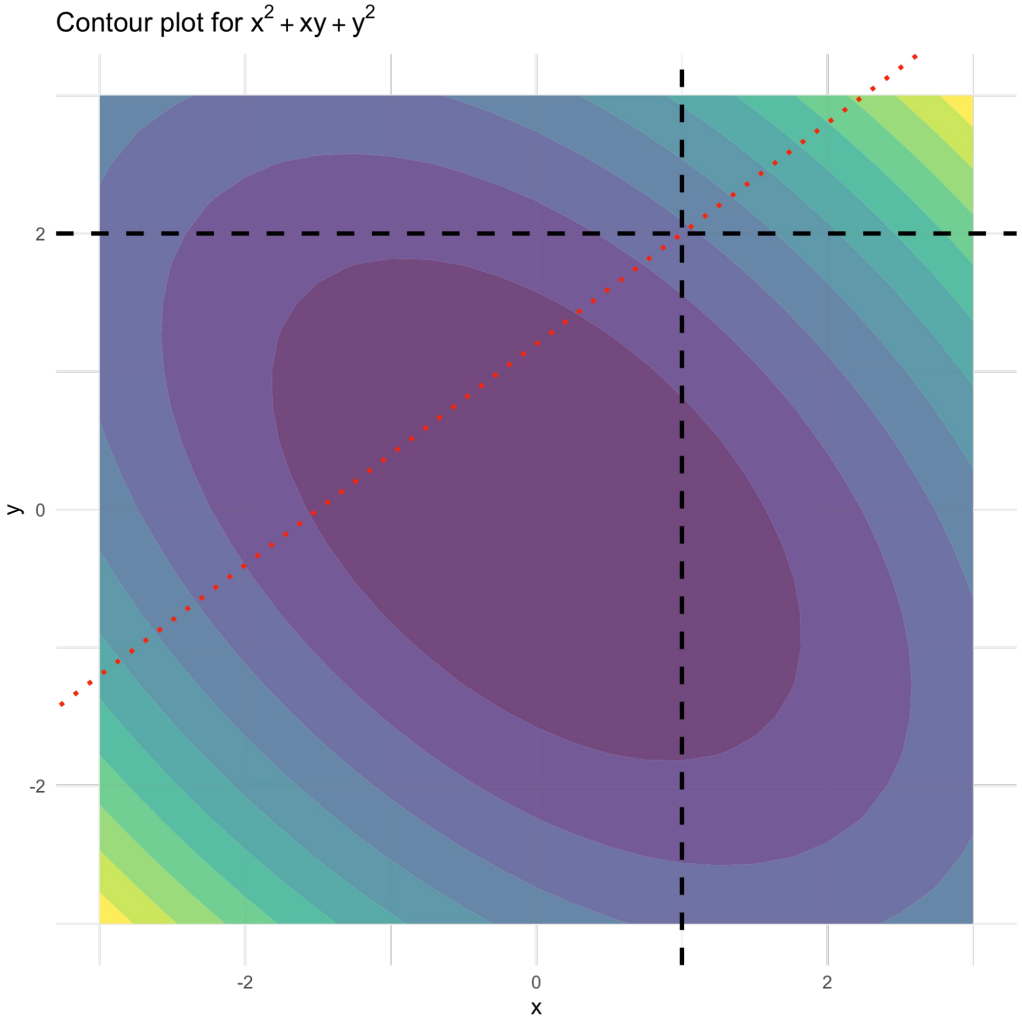
$$\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + y \\ x + 2y \end{bmatrix}$$



# Gradient

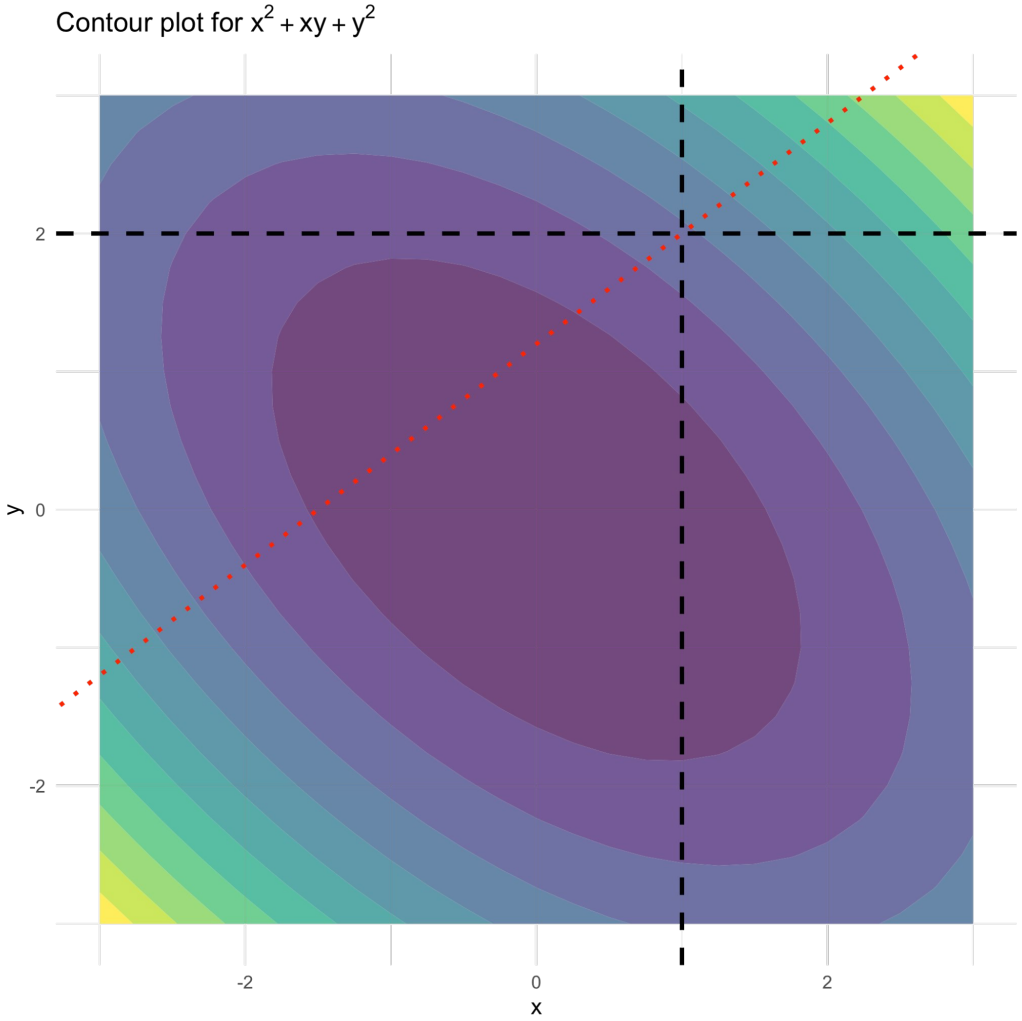
$$f = x^2 + xy + y^2$$

$$\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + y \\ x + 2y \end{bmatrix}$$



# Learning Rate

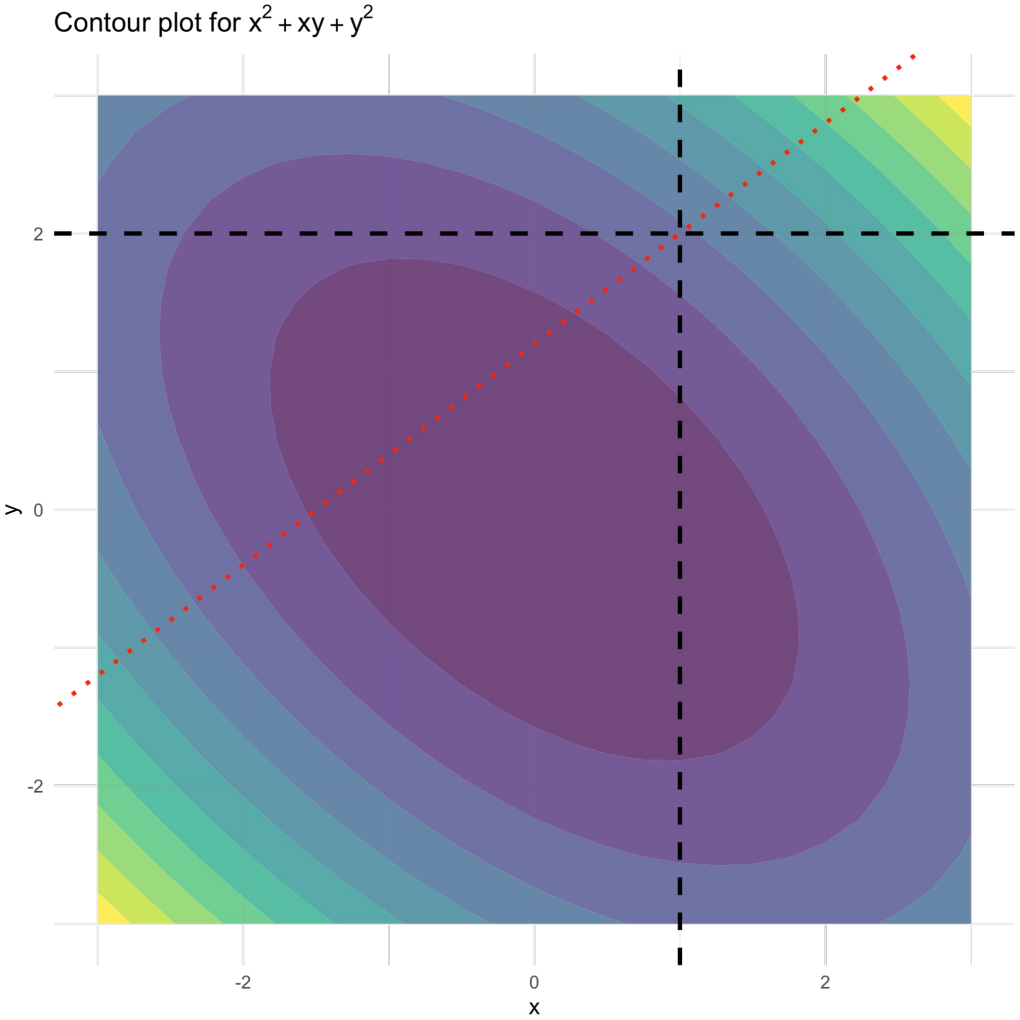
$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$



# Learning Rate

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

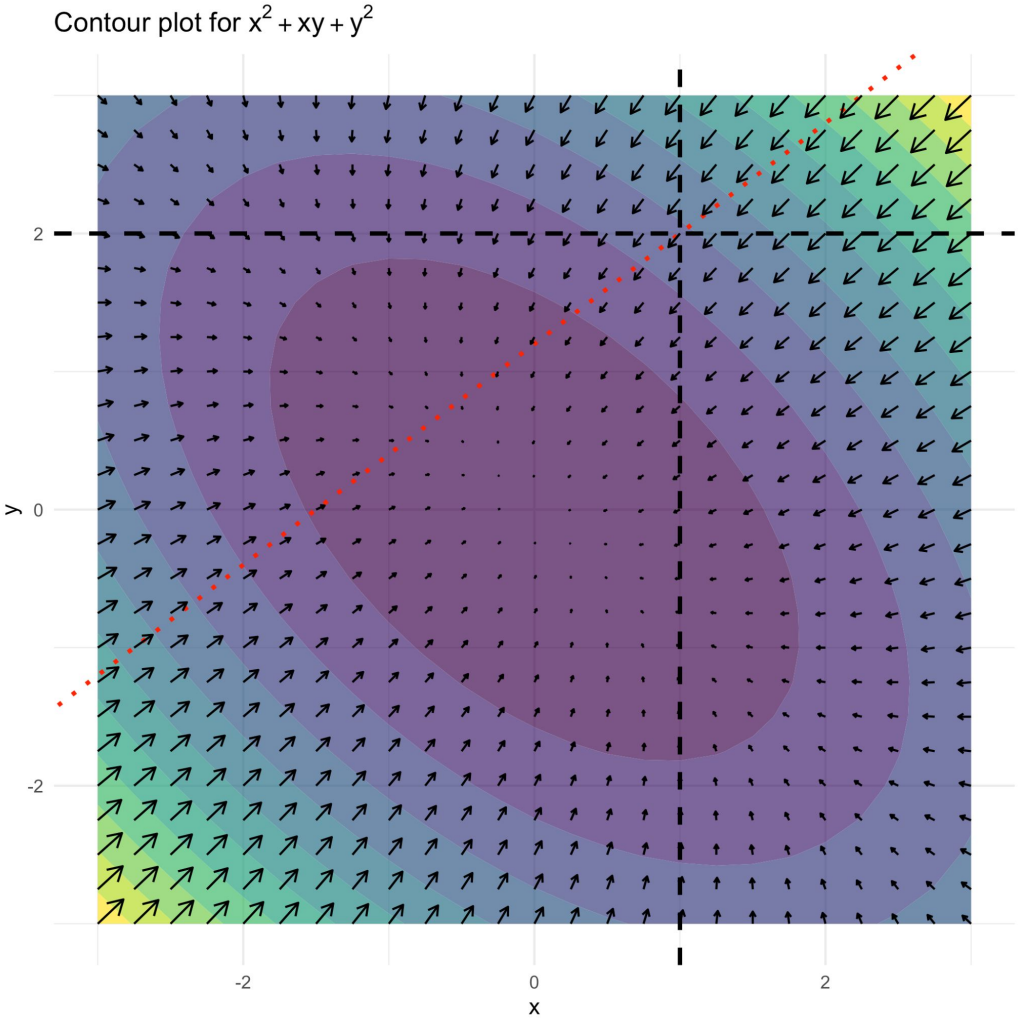
Gradients are COOL…but you can just think of it as what adjustments we should make to each of our parameters



# Learning Rate

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Gradients are COOL…but you can just think of it as what adjustments we should make to each of our parameters



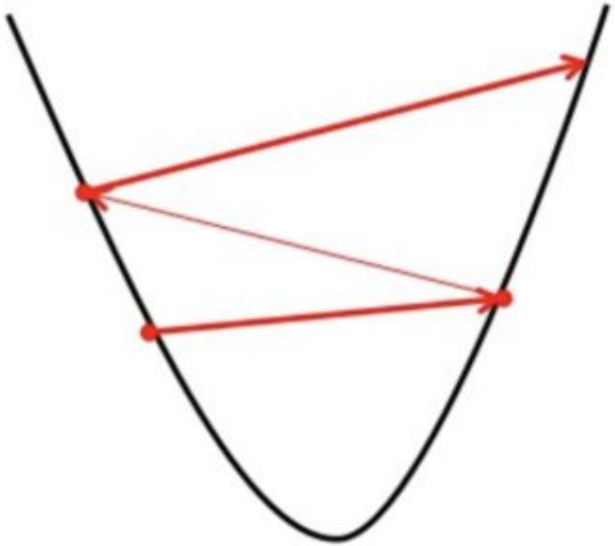
# Gradient Descent Ideas

1. The goal is to minimize the loss function
2. The gradient tells us which direction to move in
    - In other words, it tells us what adjustments to make to each parameter

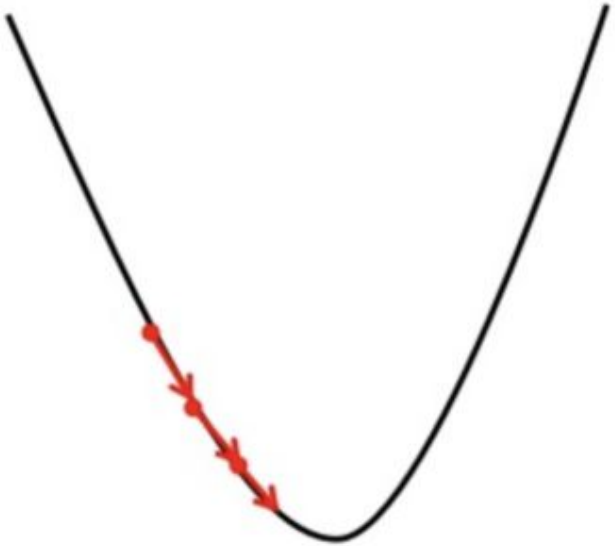
# Learning Rate

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

Big learning rate



Small learning rate



# Learning Rate Scheduler

- Constant Learning Rate: pick one learning rate for the whole process
- Learning Rate Decay: start with a larger learning rate and gradually decrease it

$$\text{learning rate} = \text{initial learning rate} * \frac{1}{(1 + decay * batch)}$$

# Gradient Descent Ideas

1. The goal is to minimize the loss function
2. The gradient tells us which direction to move in
    - In other words, it tells us what adjustments to make to each parameter
3. The Learning Rate controls how big of a step we take
    - Small steps mean slower convergence
    - Large steps mean we might step over minima

# Simple Example

$$f = x^2 + xy + y^2$$

How does **f(x,y)** change when **x** changes? What about when **y** changes?

$$\frac{\partial f}{\partial x} = 2x + y \qquad \frac{\partial f}{\partial y} = x + 2y$$

$$\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + y \\ x + 2y \end{bmatrix}$$

# Sanity Check

$$f = x^2 + xy + y^2$$

How does **f(x,y)** change when **x** changes? What about when **y** changes?

$$\frac{\partial f}{\partial x} = 2x + y \qquad \frac{\partial f}{\partial y} = x + 2y$$

$$\begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 2x + y \\ x + 2y \end{bmatrix}$$

# Linear Regression Example

$$SSE = \sum_{i}^{N} (y_i - \hat{y})^2$$

$$\hat{y} = b_0 + b_1 x$$

$$SSE = \sum_{i}^{N} (y_i - (b_0 + b_1 x))^2 = \sum_{i}^{N} (y_i - b_0 - b_1 x)^2$$

# Linear Regression Example

$$\nabla_\theta = \begin{bmatrix} \frac{\partial SSE}{\partial b_0} \\ \frac{\partial SSE}{\partial b_1} \end{bmatrix} = \begin{bmatrix} -2\sum_i^N (y_i - (b_0 + b_1 x_i)) \\ -2\sum_i^N x_i (y_i - (b_0 + b_1 x_i)) \end{bmatrix}$$

# Linear Regression Example

Let's initialize our gradient descent algorithm to start at (0,0)*

$$\begin{bmatrix} -2\sum_i^N (y_i - (0 + 0x_i)) \\ -2\sum_i^N x_i(y_i - (0 + 0x_i)) \end{bmatrix} = \begin{bmatrix} -2\sum_i^N (y_i) \\ -2\sum_i^N x_i(y_i) \end{bmatrix}$$

This makes sense, as we're guessing 0 for every point when b0 and b1 are both 0

$$\hat{y} = 0x_i + 0 = 0$$

* just because it makes our math easier

# Linear Regression Example

Notice that our gradient overall is the sum of the gradients at each of our data points

Let's pretend we have two points: (1,1) and (2,3)

$$\begin{bmatrix} -2\sum_i^N (y_i) \\ -2\sum_i^N x_i(y_i) \end{bmatrix} = \begin{bmatrix} -2(1+3) \\ -2\sum_i^N (1*1+2*3) \end{bmatrix} = \begin{bmatrix} -8 \\ -14 \end{bmatrix}$$

This shows us what changes to make to b0 and b1 in order to reduce our loss function. (Notice we should change b1 MORE than b0.

# Linear Regression Example

So let's make those changes!

$$\begin{bmatrix} b_0 \\ b_1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - 0.01 \begin{bmatrix} -8 \\ -14 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.01 \begin{bmatrix} 8 \\ 14 \end{bmatrix}$$

$$\begin{bmatrix} 0 \\ 0 \end{bmatrix} + 0.01 \begin{bmatrix} 8 \\ 14 \end{bmatrix} = \begin{bmatrix} 0.08 \\ 0.14 \end{bmatrix}$$

# Sanity Check

$$SSE = (1 - (0 + 0 * 1))^2 + (3 - 0 - 0 * 2)^2 = 1 + 9 = 10$$

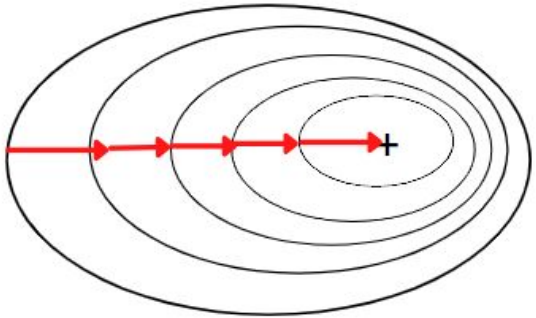
$$SSE = (1 - (0.08 + 0.14 * 1))^2 + (3 - 0.8 - 0.14 * 2)^2 = 0.6084 + 3.6864 = 4.2948$$

# Linear Regression Example

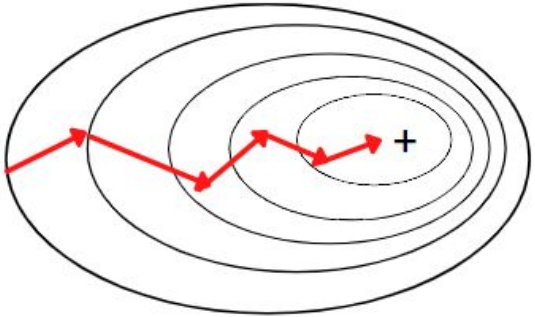
Code Example

# Flavors of GD

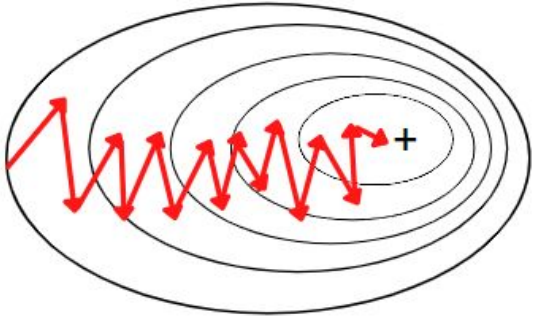
Batch Gradient Descent



Mini-Batch Gradient Descent



Stochastic Gradient Descent



# Quick Update to Notation

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} - \alpha \begin{bmatrix} \frac{\partial f}{\partial y} \\ \frac{\partial f}{\partial y} \end{bmatrix}$$

$$w_t = w_{t-1} - \alpha * g_t$$

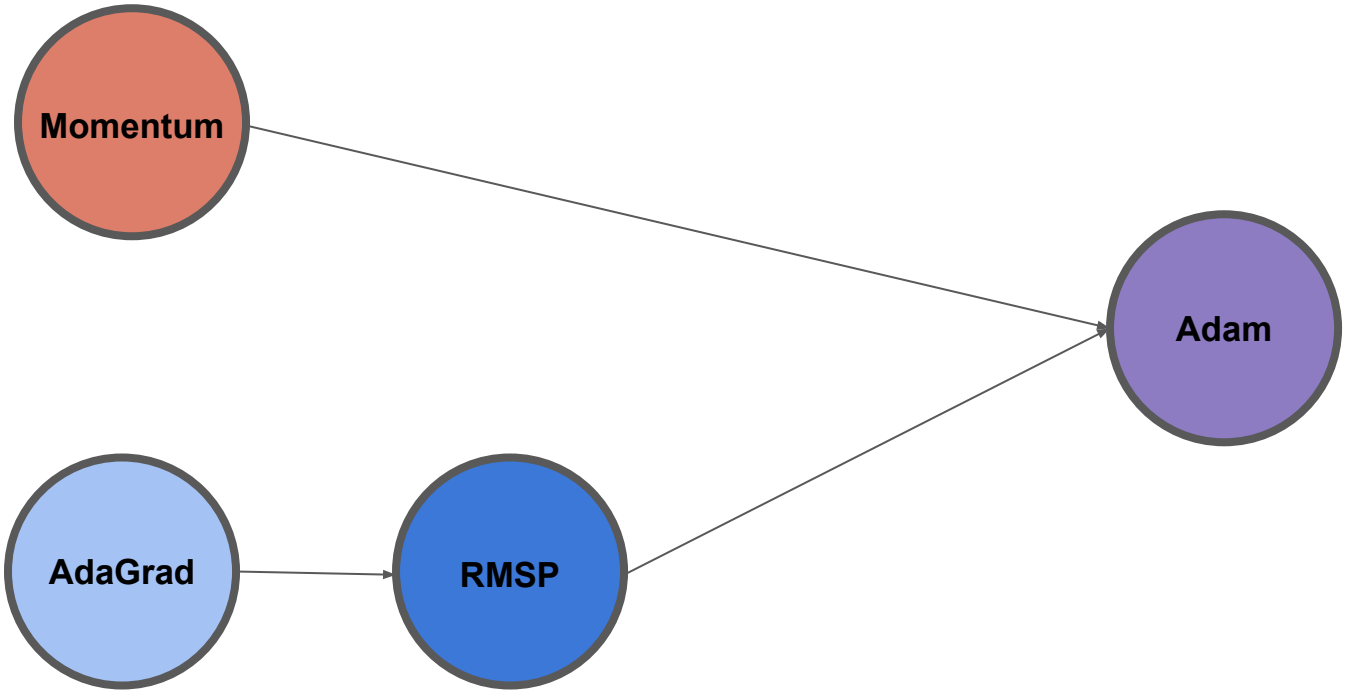
# Gradient Descent Ideas

- The Gradient uses partial derivatives to tell us how **sensitive our loss function is to changes** in each of our model's parameters
- To **update our parameters**, we take our old parameters and subtract (learning rate * gradient) $w_t = w_{t-1} - \alpha * g_t$
- **Large Learning Rates** cause us to bounce around and skip minima; **Small Learning Rates** make us SLOW
- **Stochastic** (1-data point) and **Mini-Batch** (a few data points) Gradient Descent allows us to calculate our gradient using fewer points, which is **faster**!

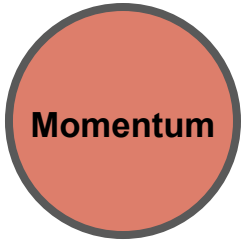
# Gradient Descent Problems

- Choosing a learning rate is hard
- Setting learning rate schedules ahead of time is hard
- Using the same learning rate for all parameters
- Local minima and saddle points

# Road Map



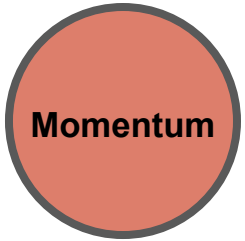
# Momentum




$$w_t = w_{t-1} - \alpha * g_t$$

What if we take into account the “momentum” we’ve gained from previous gradients?

# Momentum




$$w_t = w_{t-1} - \alpha * g_t$$

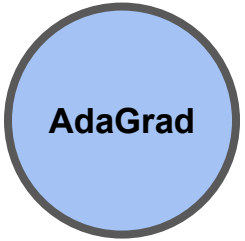
$$w_t = w_{t-1} - \alpha * m_t$$

$$m_t = \beta m_{t-1} + (1 - \beta) g_t$$

# Momentum


- Momentum allows us to “build up speed” as we go downhill
- If all of the previous steps were in the same direction, we’re more confident that we’re going in the right direction
- If all of the previous steps were in different directions, we’re less confident
- Momentum often helps us converge faster
- Momentum allows us to escape (some) local minima

# AdaGrad

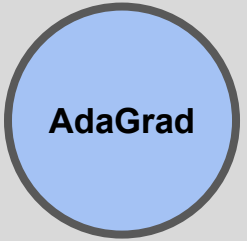



- One learning rate for each feature
- Learning rate is **ada**pted based on the sum of the squared gradients that came before

~~$w_t = w_{t-1} - \alpha * g_t$~~

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{\epsilon + \sum g_t^2}} * g_t$$

# AdaGrad




- No need to hand tune a learning rate!
- Separate learning rates for each parameter
- Step-sizes in directions with **high** gradient variance are **lower** than the step-sizes in directions with **low** gradient variance

# RMSP


- Instead of taking the sum of ALL the gradients, take an *exponential moving average* (otherwise the sum in the denominator grows continuously)

$$w_t = w_{t-1} - \alpha * g_t$$

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{\epsilon + \sum g_t^2}} * g_t$$

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{\epsilon + \nu_t}} * g_t \qquad \nu_t = \beta \nu_{t-1} + (1 - \beta) * g_t^2$$

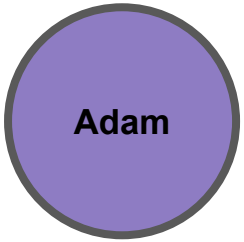
# RMSP


- AdaGrad’s learning rate gets smaller and smaller as time goes on because the denominator includes a sum of squared gradients (which can only increase)
- RMSP solves this by replacing the denominator with an exponentially decaying average of the square gradients (over time earlier gradients stop mattering)

Adam

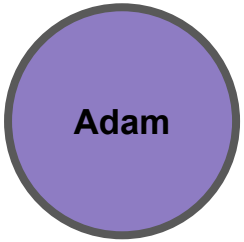

Adam

# Adam




$$w_t = w_{t-1} - \hat{m}_t\left(\frac{\alpha}{\sqrt{\hat{\nu}_t + \epsilon}}\right)$$

# Adam




$$w_t = w_{t-1} - \hat{m}_t\left(\frac{\alpha}{\sqrt{\hat{\nu}_t + \epsilon}}\right)$$

RMSP

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{\epsilon + \nu_t}} * g_t$$

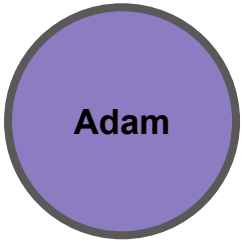
$$\nu_t = \beta\nu_{t-1} + (1-\beta) * g_t^2$$

Momentum

$$w_t = w_{t-1} - \alpha * m_t$$

$$m_t = \beta m_{t-1} + (1-\beta)g_t$$

# Adam




$$w_t = w_{t-1} - \hat{m}_t\left(\frac{\alpha}{\sqrt{\hat{\nu}_t + \epsilon}}\right)$$

$$\hat{\nu}_t = \frac{\nu_t}{1-\beta_2^t} \quad \hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

A way to unbias the parameters

$$w_t = w_{t-1} - \frac{\alpha}{\sqrt{\epsilon + \nu_t}} * g_t$$

$$\nu_t = \beta\nu_{t-1} + (1-\beta) * g_t^2$$

$$w_t = w_{t-1} - \alpha * m_t$$

$$m_t = \beta m_{t-1} + (1-\beta) g_t$$