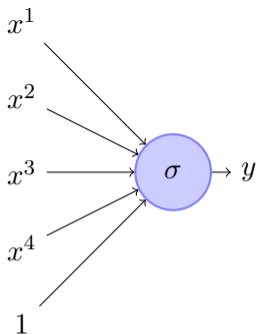


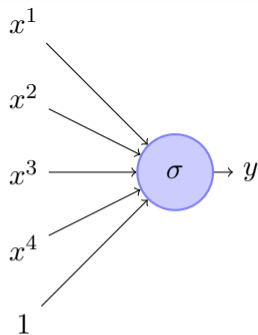
## Module 5.9 : Gradient Descent with Adaptive Learning Rate



$$y = f(x) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

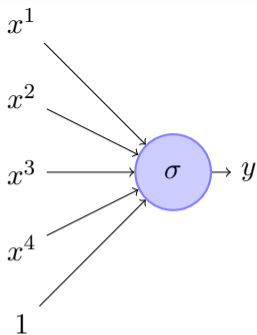


- Given this network, it should be easy to see that given a single point  $(\mathbf{x}, y)$ ...

$$y = f(x) = \frac{1}{1+e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

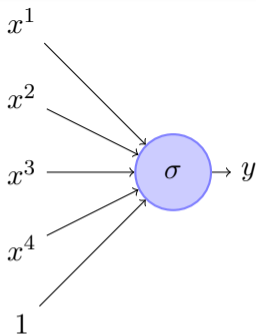


- Given this network, it should be easy to see that given a single point  $(\mathbf{x}, y)$ ...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$  ... so on

$$y = f(x) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

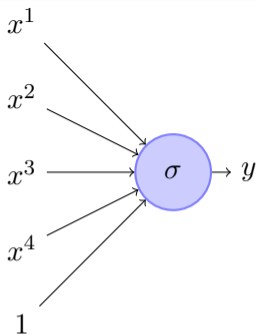


- Given this network, it should be easy to see that given a single point  $(\mathbf{x}, y)$ ...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$  ... so on
- If there are  $n$  points, we can just sum the gradients over all the  $n$  points to get the total gradient

$$y = f(x) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

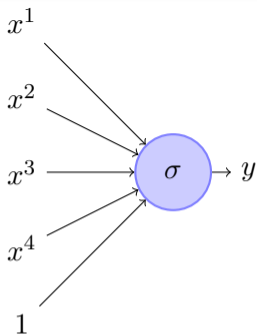


- Given this network, it should be easy to see that given a single point  $(\mathbf{x}, y)$ ...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$  ... so on
- If there are  $n$  points, we can just sum the gradients over all the  $n$  points to get the total gradient
- What happens if the feature  $x^2$  is very sparse?

$$y = f(x) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

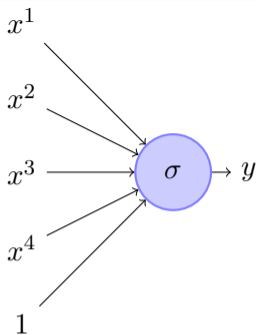


- Given this network, it should be easy to see that given a single point  $(\mathbf{x}, y)$ ...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$  ... so on
- If there are  $n$  points, we can just sum the gradients over all the  $n$  points to get the total gradient
- What happens if the feature  $x^2$  is very sparse? (*i.e.*, if its value is 0 for most inputs)

$$y = f(x) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

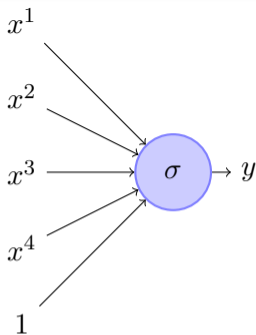


$$y = f(x) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point  $(\mathbf{x}, y)$ ...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2 \dots$  so on
- If there are  $n$  points, we can just sum the gradients over all the  $n$  points to get the total gradient
- What happens if the feature  $x^2$  is very sparse? (*i.e.*, if its value is 0 for most inputs)
- $\nabla w^2$  will be 0 for most inputs (see formula) and hence  $w^2$  will not get enough updates

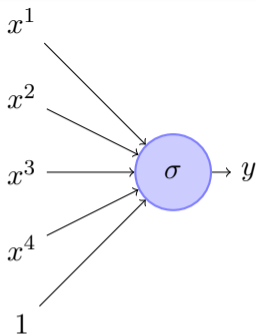


$$y = f(x) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point  $(\mathbf{x}, y)$ ...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$  ... so on
- If there are  $n$  points, we can just sum the gradients over all the  $n$  points to get the total gradient
- What happens if the feature  $x^2$  is very sparse? (*i.e.*, if its value is 0 for most inputs)
- $\nabla w^2$  will be 0 for most inputs (see formula) and hence  $w^2$  will not get enough updates
- If  $x^2$  happens to be sparse as well as important we would want to take the updates to  $w^2$  more seriously



$$y = f(x) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}$$

$$\mathbf{x} = \{x^1, x^2, x^3, x^4\}$$

$$\mathbf{w} = \{w^1, w^2, w^3, w^4\}$$

- Given this network, it should be easy to see that given a single point  $(\mathbf{x}, y)$ ...
- $\nabla w^1 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^1$
- $\nabla w^2 = (f(\mathbf{x}) - y) * f(\mathbf{x}) * (1 - f(\mathbf{x})) * x^2$  ... so on
- If there are  $n$  points, we can just sum the gradients over all the  $n$  points to get the total gradient
- What happens if the feature  $x^2$  is very sparse? (*i.e.*, if its value is 0 for most inputs)
- $\nabla w^2$  will be 0 for most inputs (see formula) and hence  $w^2$  will not get enough updates
- If  $x^2$  happens to be sparse as well as important we would want to take the updates to  $w^2$  more seriously
- Can we have a different learning rate for each parameter which takes care of the frequency of features ?

## Intuition

- Decay the learning rate for parameters in proportion to their update history

## Intuition

- Decay the learning rate for parameters in proportion to their update history (more updates means more decay)

## Intuition

- Decay the learning rate for parameters in proportion to their update history (more updates means more decay)

## Update rule for Adagrad

$$v_t = v_{t-1} + (\nabla w_t)^2$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * \nabla w_t$$

... and a similar set of equations for  $b_t$

- To see this in action we need to first create some data where one of the features is sparse

```
def do_adagrad():  
    w, b, eta = init_w, init_b, 0.1  
    v_w, v_b, eps = 0, 0, 1e-8  
    for i in range(max_epochs):  
        dw, db = 0, 0  
        for x,y in zip(X, Y):  
            dw += grad_w(w, b, x, y)  
            db += grad_b(w, b, x, y)  
  
        v_w = v_w + dw**2  
        v_b = v_b + db**2  
  
        w = w - (eta / np.sqrt(v_w + eps)) * dw  
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ?

```
def do_adagrad():  
    w, b, eta = init_w, init_b, 0.1  
    v_w, v_b, eps = 0, 0, 1e-8  
    for i in range(max_epochs):  
        dw, db = 0, 0  
        for x,y in zip(X, Y):  
            dw += grad_w(w, b, x, y)  
            db += grad_b(w, b, x, y)  
  
        v_w = v_w + dw**2  
        v_b = v_b + db**2  
  
        w = w - (eta / np.sqrt(v_w + eps)) * dw  
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ?  
Take some time to think about it

```
def do_adagrad():  
    w, b, eta = init_w, init_b, 0.1  
    v_w, v_b, eps = 0, 0, 1e-8  
    for i in range(max_epochs):  
        dw, db = 0, 0  
        for x,y in zip(X, Y):  
            dw += grad_w(w, b, x, y)  
            db += grad_b(w, b, x, y)  
  
        v_w = v_w + dw**2  
        v_b = v_b + db**2  
  
        w = w - (eta / np.sqrt(v_w + eps)) * dw  
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ?  
Take some time to think about it
- Well, our network has just two parameters ( $w$  and  $b$ ).

```
def do_adagrad():  
    w, b, eta = init_w, init_b, 0.1  
    v_w, v_b, eps = 0, 0, 1e-8  
    for i in range(max_epochs):  
        dw, db = 0, 0  
        for x,y in zip(X, Y):  
            dw += grad_w(w, b, x, y)  
            db += grad_b(w, b, x, y)  
  
        v_w = v_w + dw**2  
        v_b = v_b + db**2  
  
        w = w - (eta / np.sqrt(v_w + eps)) * dw  
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ?  
Take some time to think about it
- Well, our network has just two parameters ( $w$  and  $b$ ). Of these, the input/feature corresponding to  $b$  is always on

```
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ?  
Take some time to think about it
- Well, our network has just two parameters ( $w$  and  $b$ ). Of these, the input/feature corresponding to  $b$  is always on (so can't really make it sparse)

```
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ?  
Take some time to think about it
- Well, our network has just two parameters ( $w$  and  $b$ ). Of these, the input/feature corresponding to  $b$  is always on (so can't really make it sparse)
- The only option is to make  $x$  sparse

```
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ?  
Take some time to think about it
- Well, our network has just two parameters ( $w$  and  $b$ ). Of these, the input/feature corresponding to  $b$  is always on (so can't really make it sparse)
- The only option is to make  $x$  sparse
- **Solution:** We created 100 random  $(x, y)$  pairs and then for roughly 80% of these pairs we set  $x$  to 0

```
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

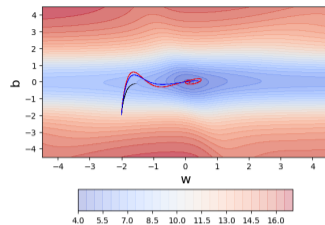
- To see this in action we need to first create some data where one of the features is sparse
- How would we do this in our toy network ? Take some time to think about it
- Well, our network has just two parameters ( $w$  and  $b$ ). Of these, the input/feature corresponding to  $b$  is always on (so can't really make it sparse)
- The only option is to make  $x$  sparse
- **Solution:** We created 100 random  $(x, y)$  pairs and then for roughly 80% of these pairs we set  $x$  to 0 thereby, making the feature for  $w$  sparse

```
def do_adagrad():
    w, b, eta = init_w, init_b, 0.1
    v_w, v_b, eps = 0, 0, 1e-8
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

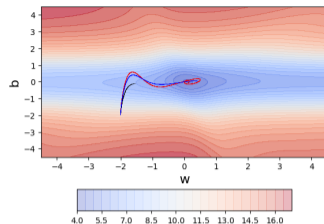
        v_w = v_w + dw**2
        v_b = v_b + db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

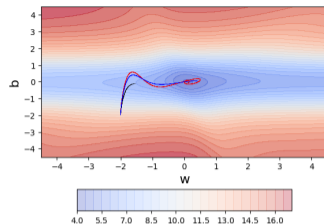
- GD (black), momentum (red) and NAG (blue)



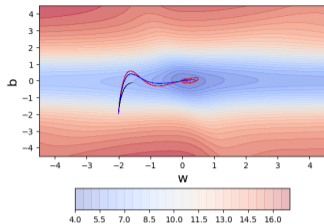
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset.



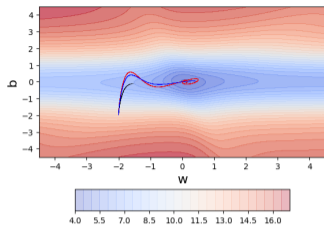
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?



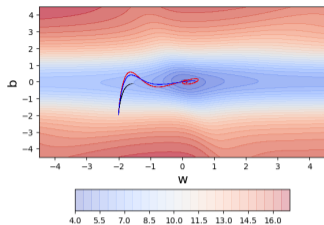
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical ( $b$ ) axis and there is very little movement along the horizontal ( $w$ ) axis



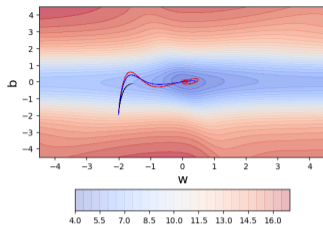
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical ( $b$ ) axis and there is very little movement along the horizontal ( $w$ ) axis
- Why?



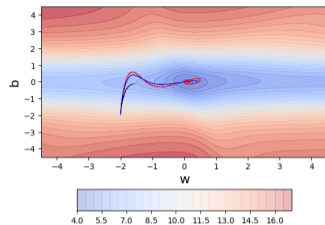
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical ( $b$ ) axis and there is very little movement along the horizontal ( $w$ ) axis
- Why? Because in our data, the feature corresponding to  $w$  is sparse and hence  $w$  undergoes very few updates



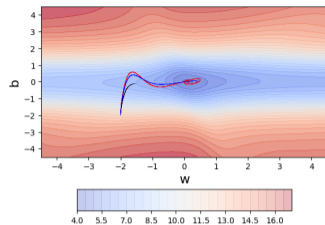
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical ( $b$ ) axis and there is very little movement along the horizontal ( $w$ ) axis
- Why? Because in our data, the feature corresponding to  $w$  is sparse and hence  $w$  undergoes very few updates ...on the other hand  $b$  is very dense and undergoes many updates



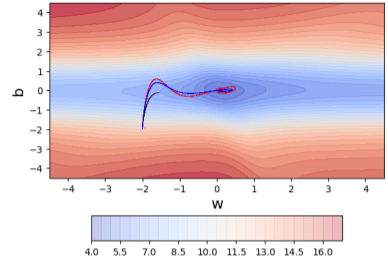
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical ( $b$ ) axis and there is very little movement along the horizontal ( $w$ ) axis
- Why? Because in our data, the feature corresponding to  $w$  is sparse and hence  $w$  undergoes very few updates ...on the other hand  $b$  is very dense and undergoes many updates
- Such sparsity is very common in large neural networks containing 1000s of input features and hence we need to address it

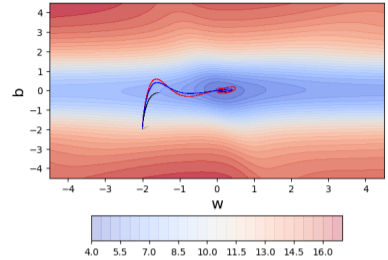


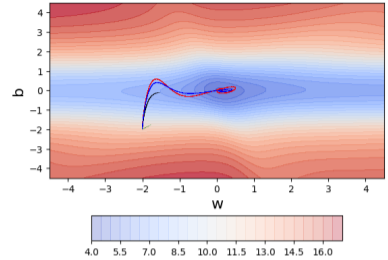
- GD (black), momentum (red) and NAG (blue)
- There is something interesting that these 3 algorithms are doing for this dataset. Can you spot it?
- Initially, all three algorithms are moving mainly along the vertical ( $b$ ) axis and there is very little movement along the horizontal ( $w$ ) axis
- Why? Because in our data, the feature corresponding to  $w$  is sparse and hence  $w$  undergoes very few updates ...on the other hand  $b$  is very dense and undergoes many updates
- Such sparsity is very common in large neural networks containing 1000s of input features and hence we need to address it

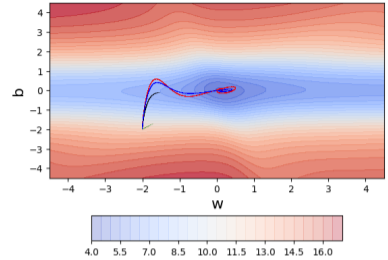


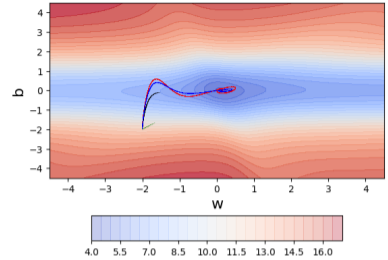
- Let's see what Adagrad does....

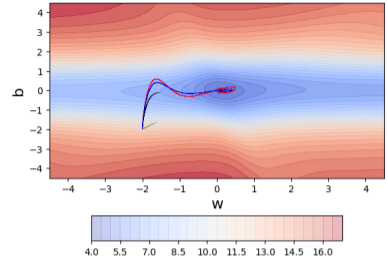


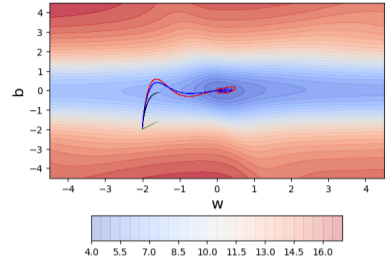


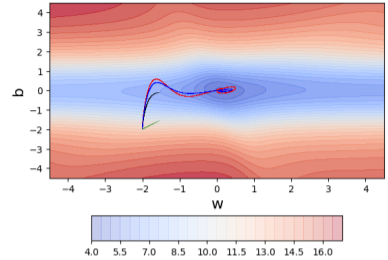


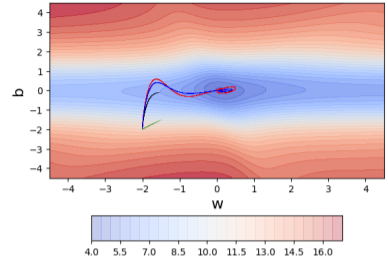


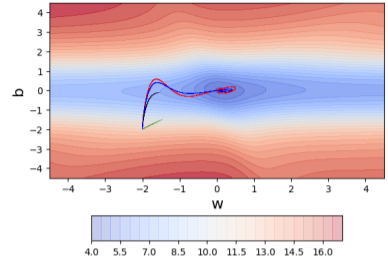


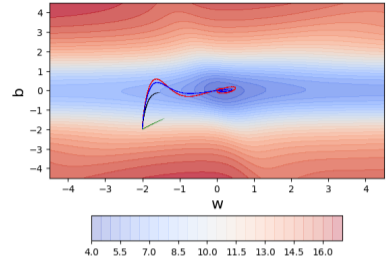


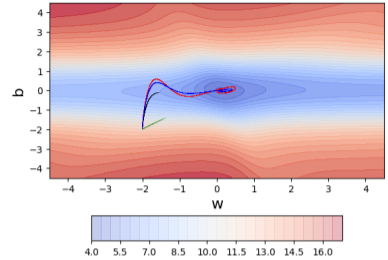


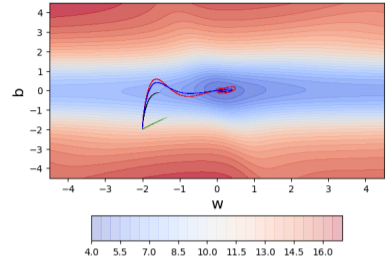


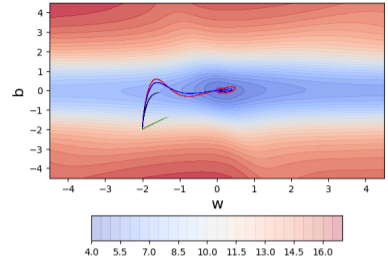


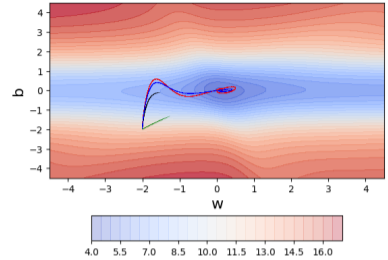


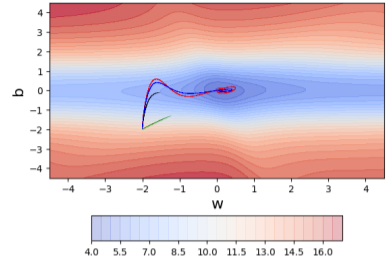


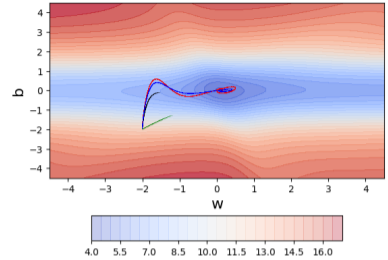


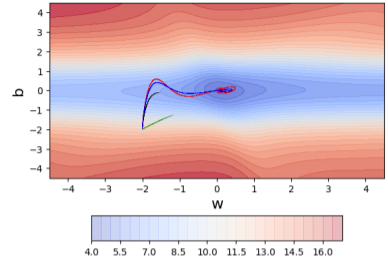


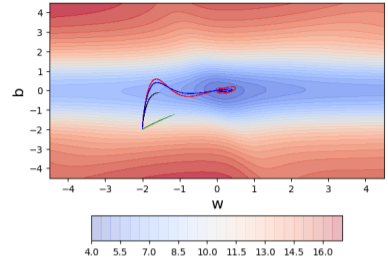


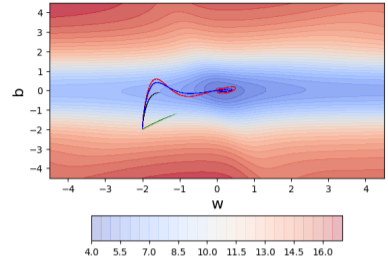


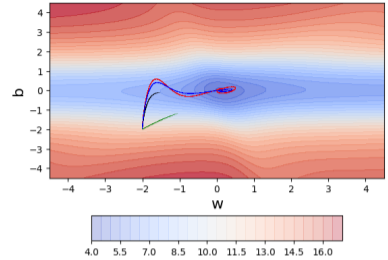


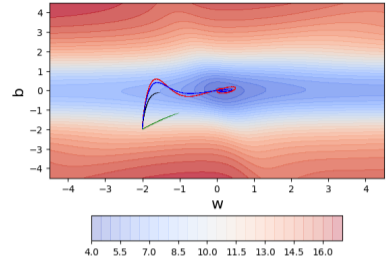


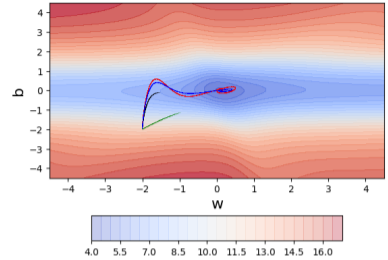


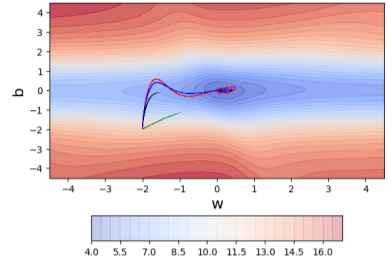


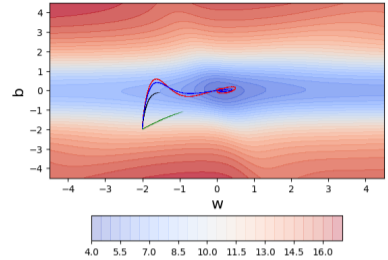


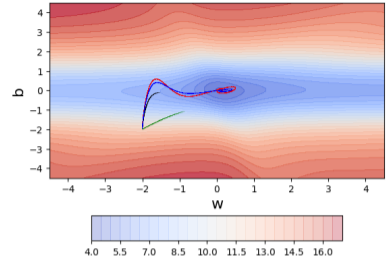


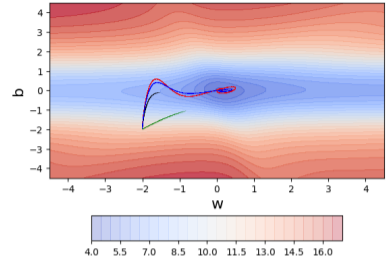


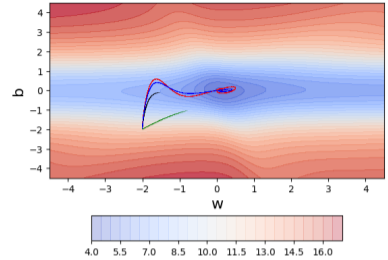


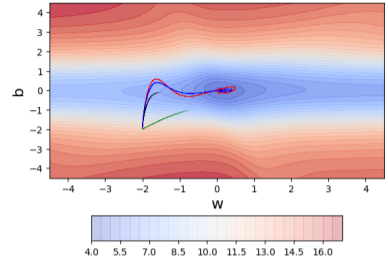


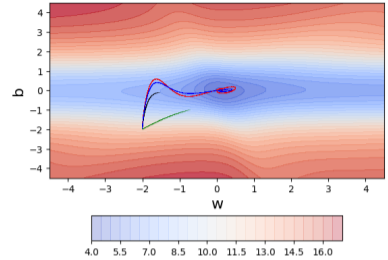


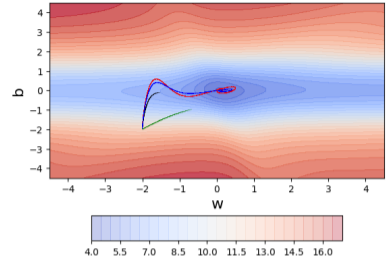


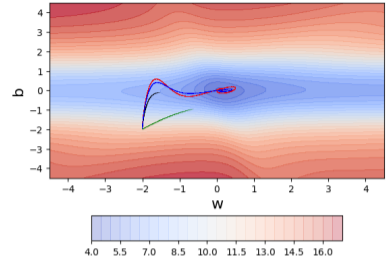


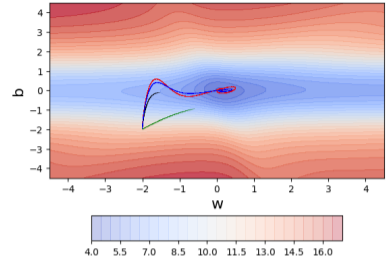


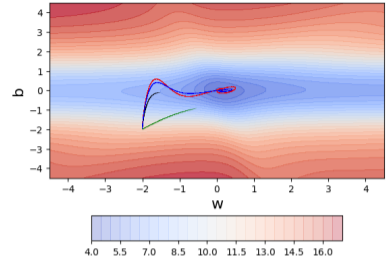


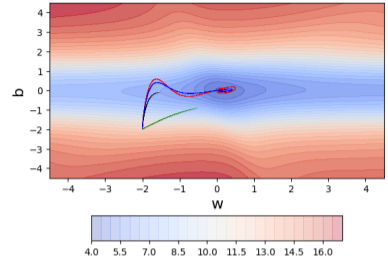


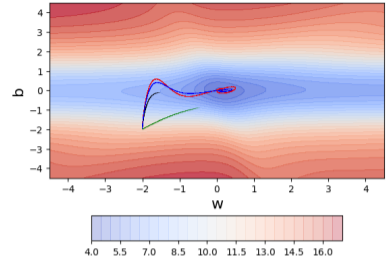


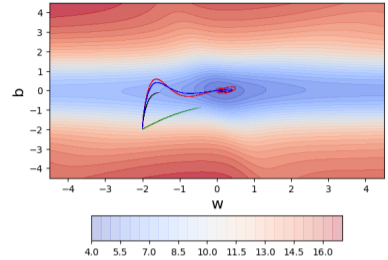


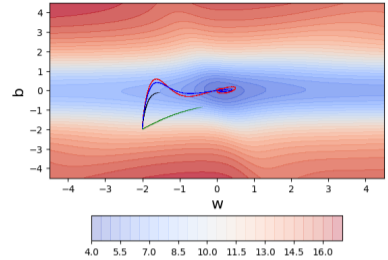


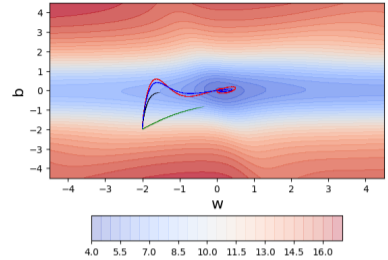


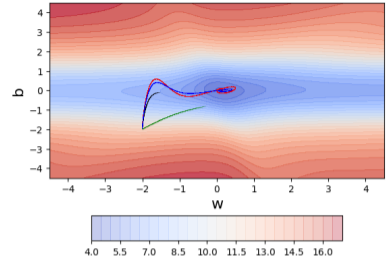


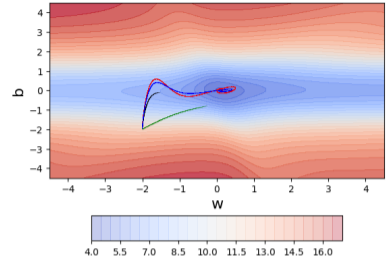


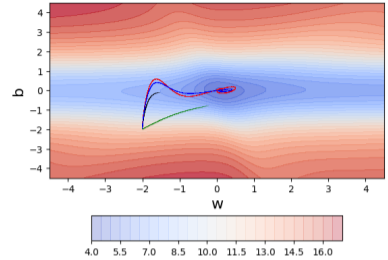


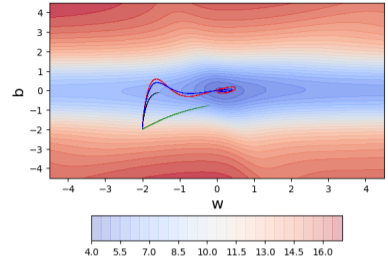


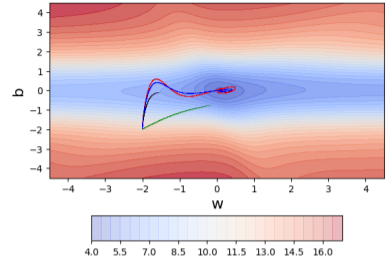


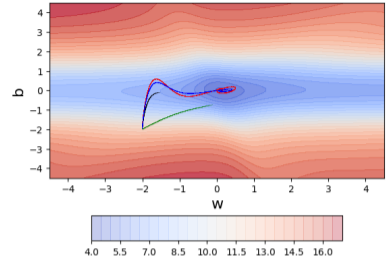


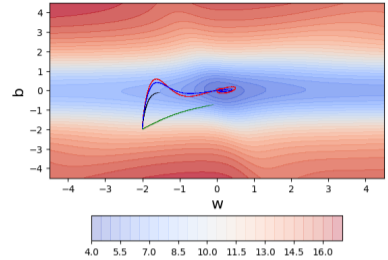


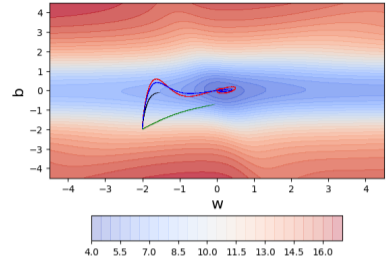


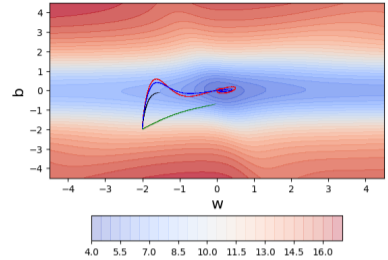


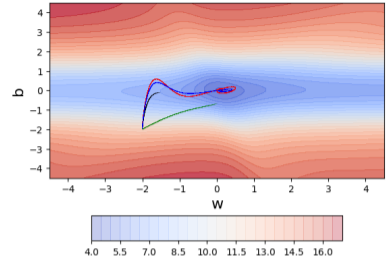


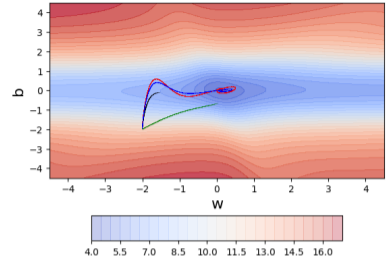


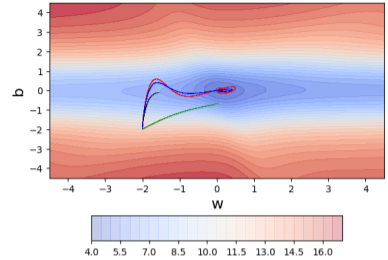


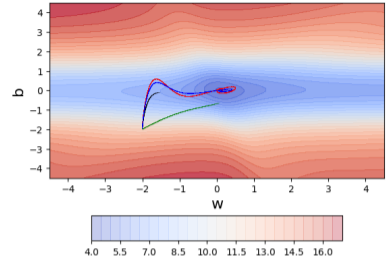


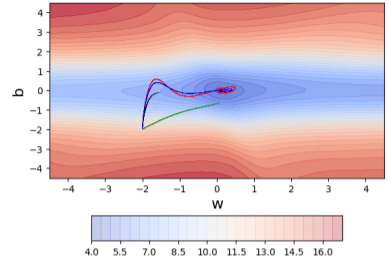


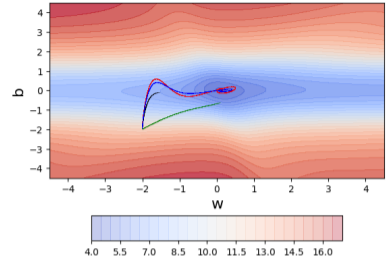


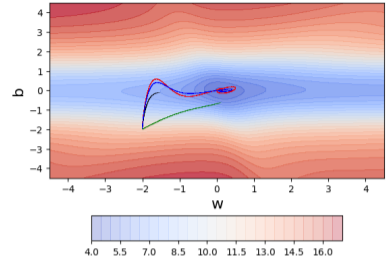


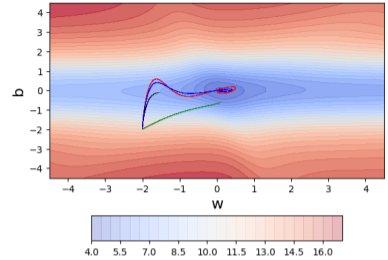


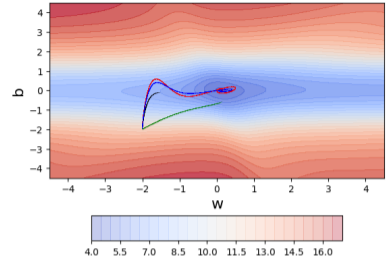


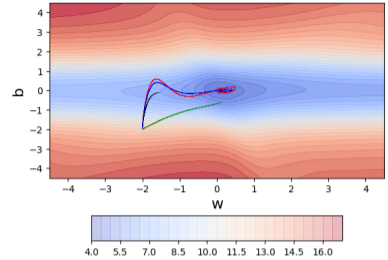


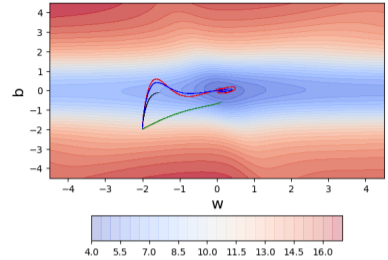


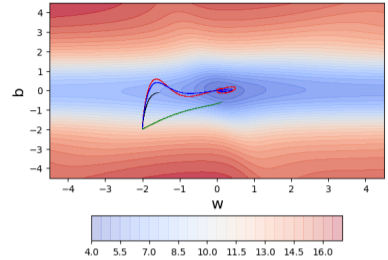


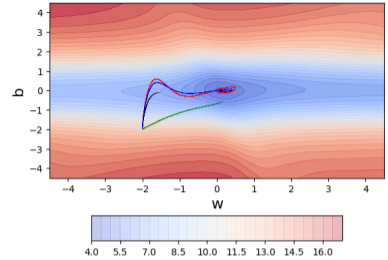


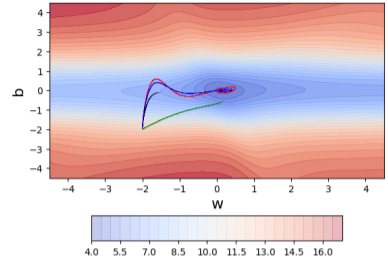


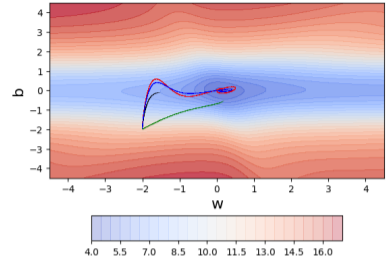


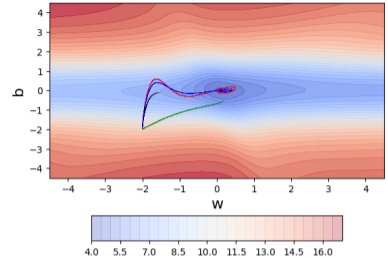


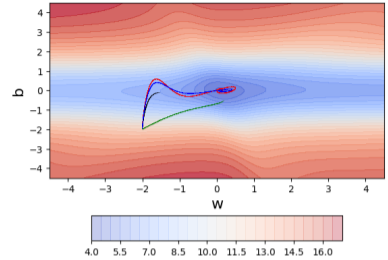


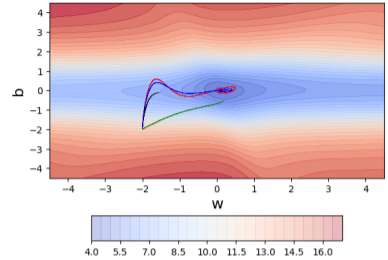


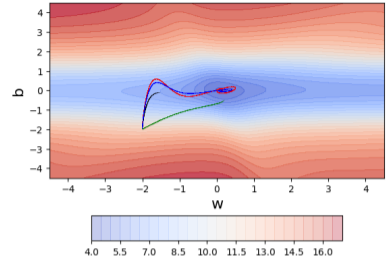


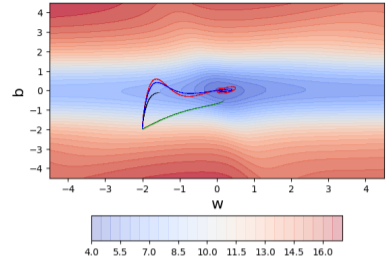


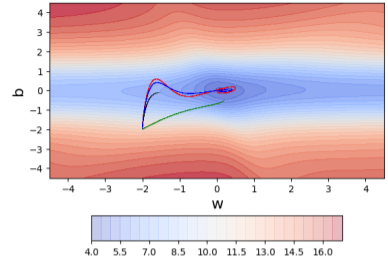


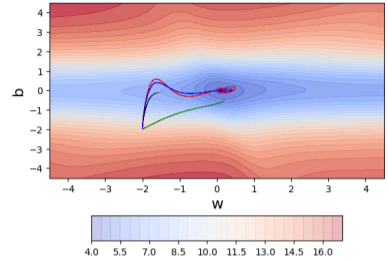


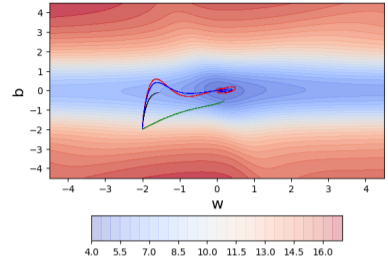


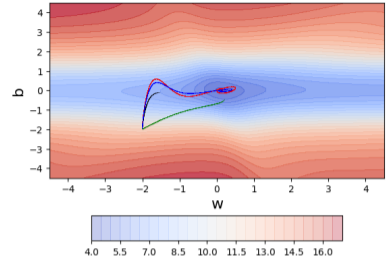


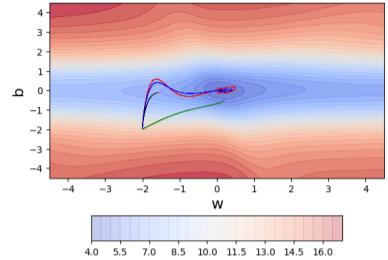


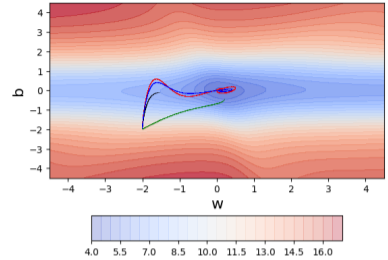


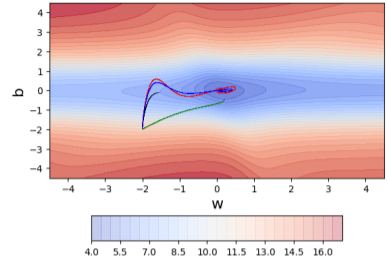


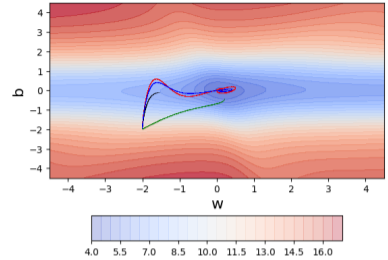


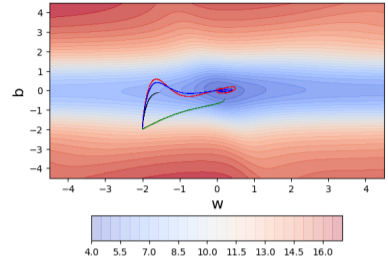


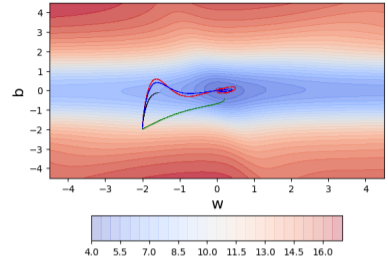


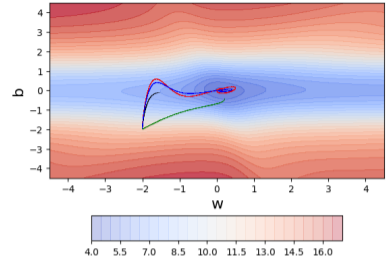


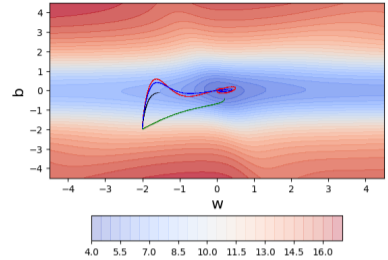


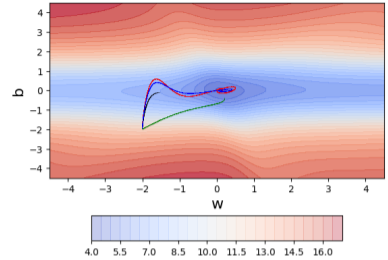


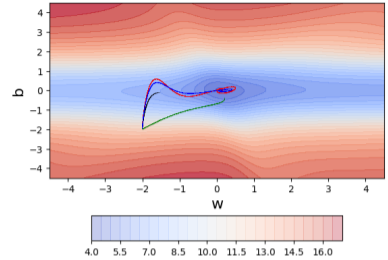


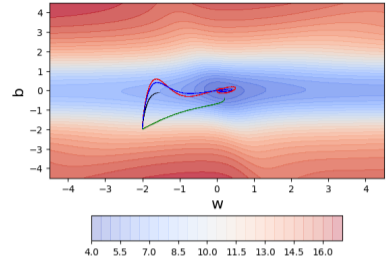


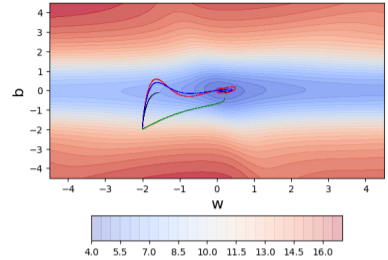


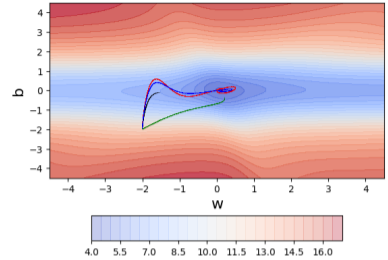


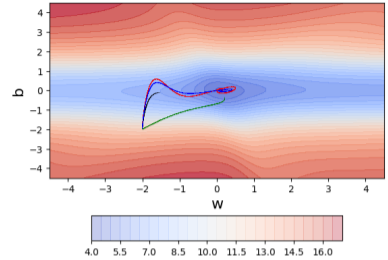


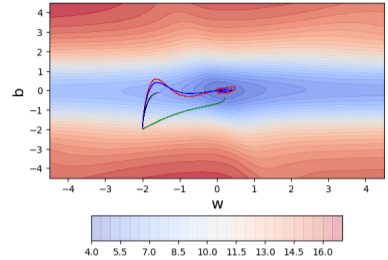


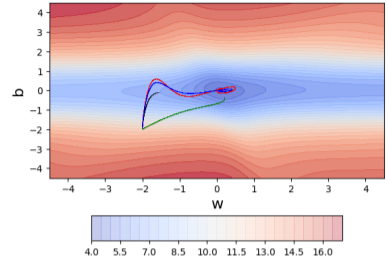


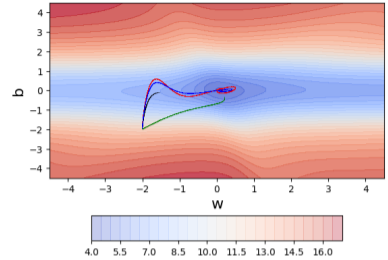


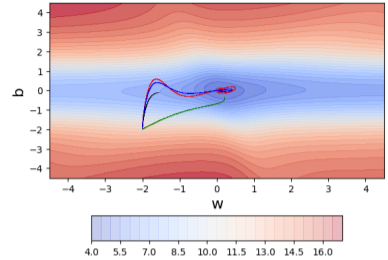




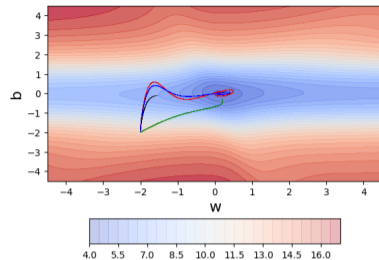




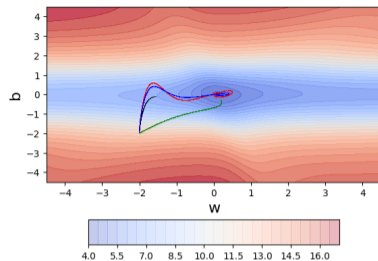




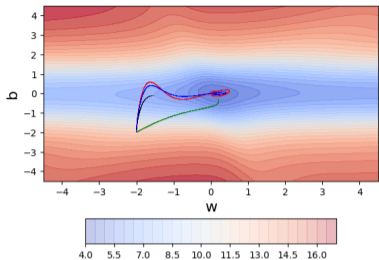
- By using a parameter specific learning rate it ensures that despite sparsity  $w$  gets a higher learning rate and hence larger updates



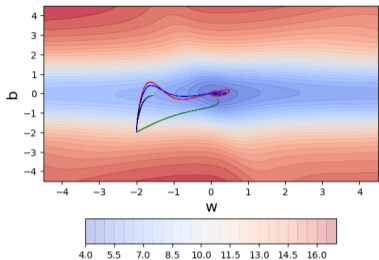
- By using a parameter specific learning rate it ensures that despite sparsity  $w$  gets a higher learning rate and hence larger updates
- Further, it also ensures that if  $b$  undergoes a lot of updates its effective learning rate decreases because of the growing denominator



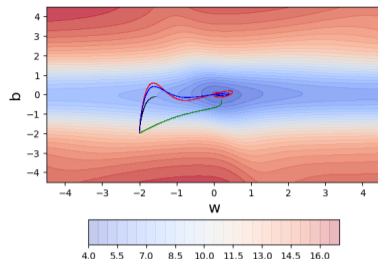
- By using a parameter specific learning rate it ensures that despite sparsity  $w$  gets a higher learning rate and hence larger updates
- Further, it also ensures that if  $b$  undergoes a lot of updates its effective learning rate decreases because of the growing denominator
- In practice, this does not work so well if we remove the square root from the denominator



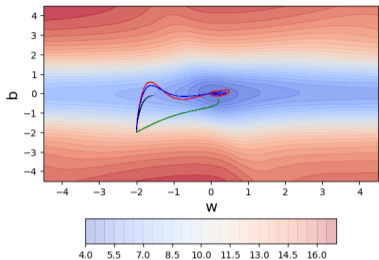
- By using a parameter specific learning rate it ensures that despite sparsity  $w$  gets a higher learning rate and hence larger updates
- Further, it also ensures that if  $b$  undergoes a lot of updates its effective learning rate decreases because of the growing denominator
- In practice, this does not work so well if we remove the square root from the denominator (something to ponder about)



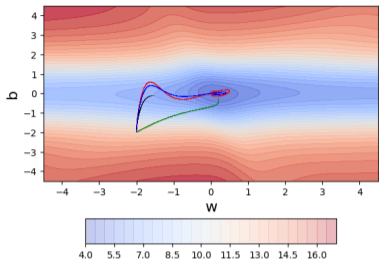
- By using a parameter specific learning rate it ensures that despite sparsity  $w$  gets a higher learning rate and hence larger updates
- Further, it also ensures that if  $b$  undergoes a lot of updates its effective learning rate decreases because of the growing denominator
- In practice, this does not work so well if we remove the square root from the denominator (something to ponder about)
- What's the flipside?



- By using a parameter specific learning rate it ensures that despite sparsity  $w$  gets a higher learning rate and hence larger updates
- Further, it also ensures that if  $b$  undergoes a lot of updates its effective learning rate decreases because of the growing denominator
- In practice, this does not work so well if we remove the square root from the denominator (something to ponder about)
- What's the flipside? over time the effective learning rate for  $b$  will decay to an extent that there will be no further updates to  $b$



- By using a parameter specific learning rate it ensures that despite sparsity  $w$  gets a higher learning rate and hence larger updates
- Further, it also ensures that if  $b$  undergoes a lot of updates its effective learning rate decreases because of the growing denominator
- In practice, this does not work so well if we remove the square root from the denominator (something to ponder about)
- What's the flipside? over time the effective learning rate for  $b$  will decay to an extent that there will be no further updates to  $b$
- Can we avoid this?



## Intuition

- Adagrad decays the learning rate very aggressively (as the denominator grows)

## Intuition

- Adagrad decays the learning rate very aggressively (as the denominator grows)
- As a result after a while the frequent parameters will start receiving very small updates because of the decayed learning rate

## Intuition

- Adagrad decays the learning rate very aggressively (as the denominator grows)
- As a result after a while the frequent parameters will start receiving very small updates because of the decayed learning rate
- To avoid this why not decay the denominator and prevent its rapid growth

## Intuition

- Adagrad decays the learning rate very aggressively (as the denominator grows)
- As a result after a while the frequent parameters will start receiving very small updates because of the decayed learning rate
- To avoid this why not decay the denominator and prevent its rapid growth

## Update rule for RMSProp

$$v_t = \beta * v_{t-1} + (1 - \beta)(\nabla w_t)^2$$
$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t + \epsilon}} * \nabla w_t$$

... and a similar set of equations for  $b_t$

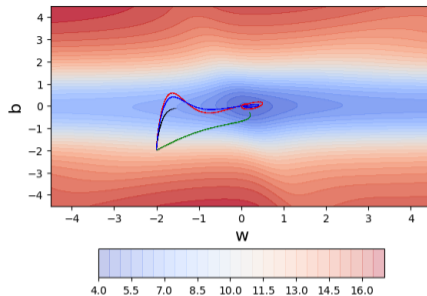
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



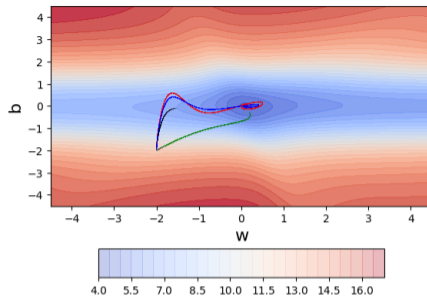
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



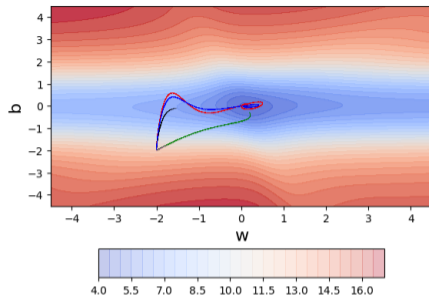
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



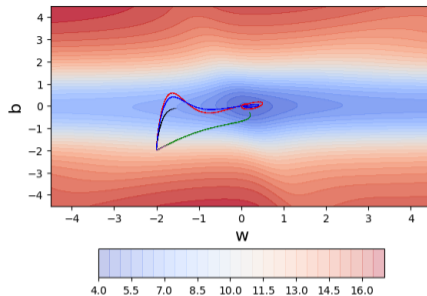
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



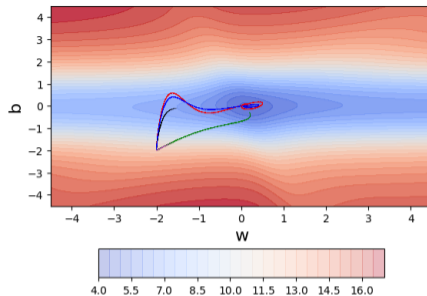
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



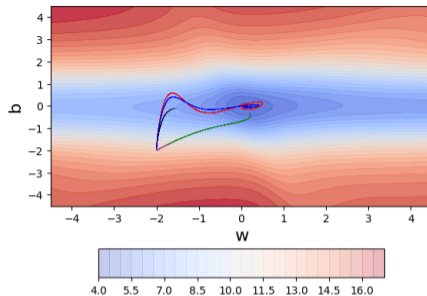
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



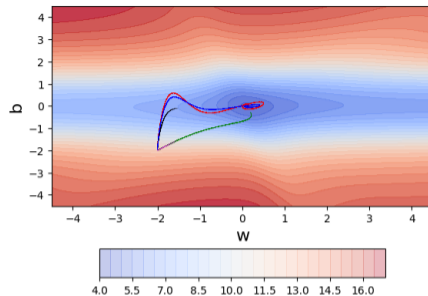
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



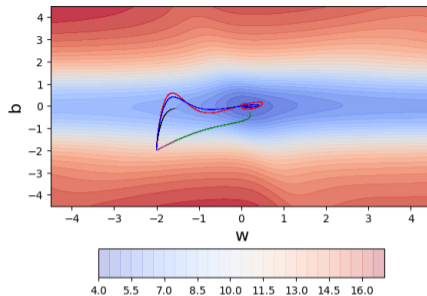
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



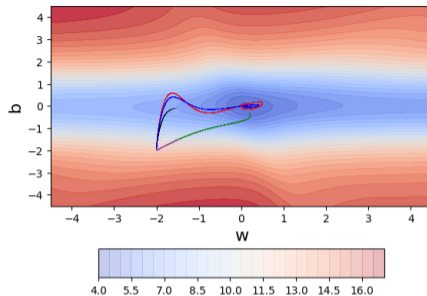
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



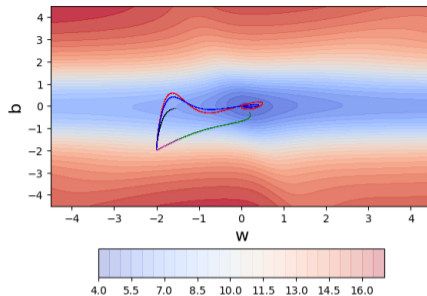
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



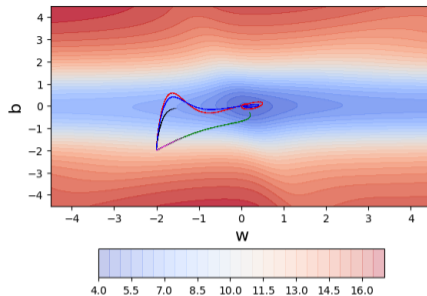
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



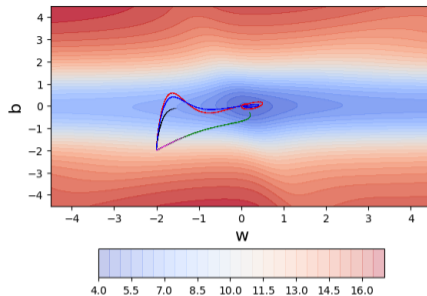
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



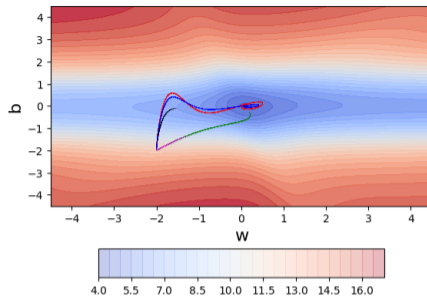
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



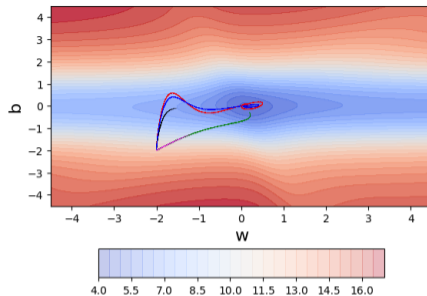
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



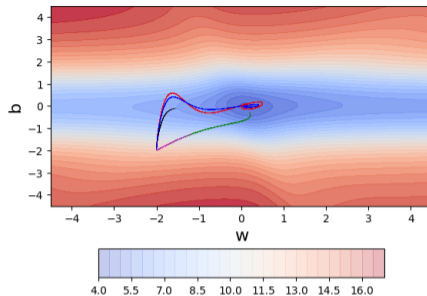
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



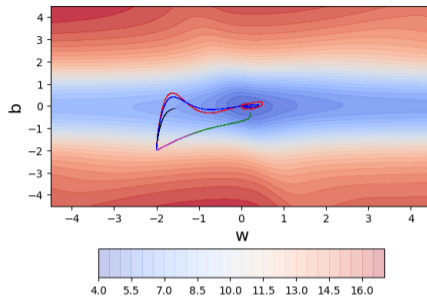
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



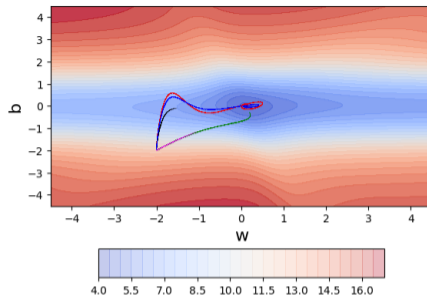
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



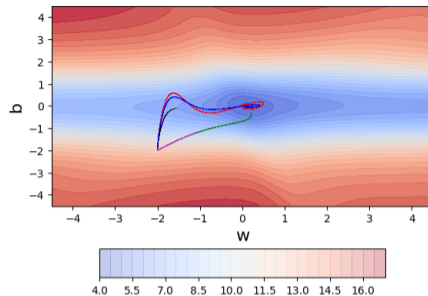
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



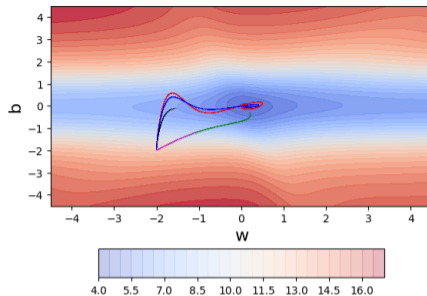
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



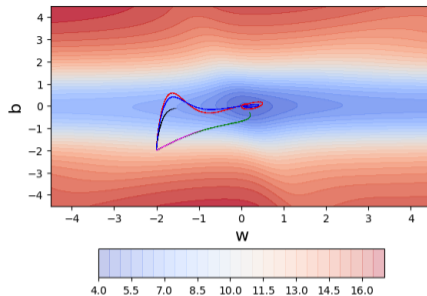
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



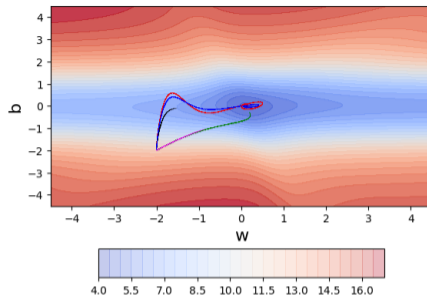
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



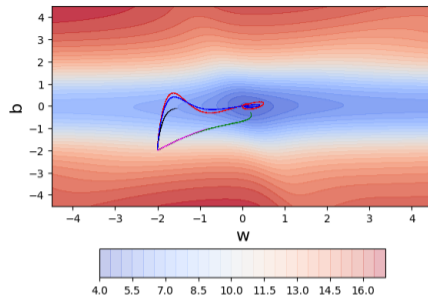
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



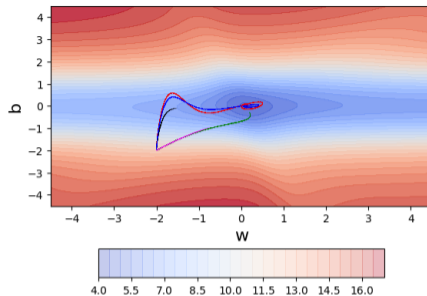
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



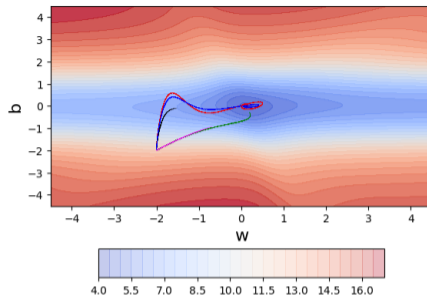
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



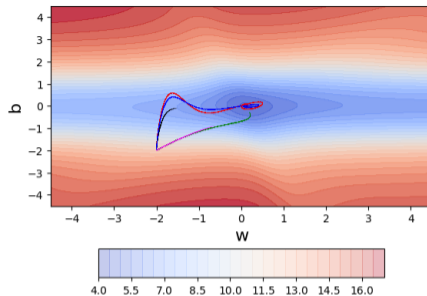
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



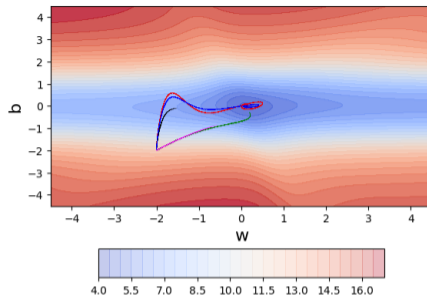
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



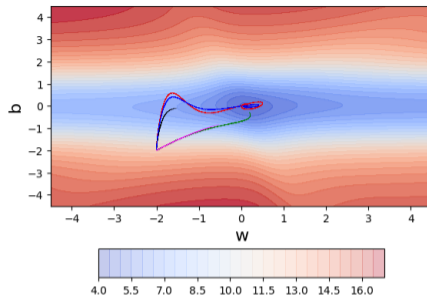
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



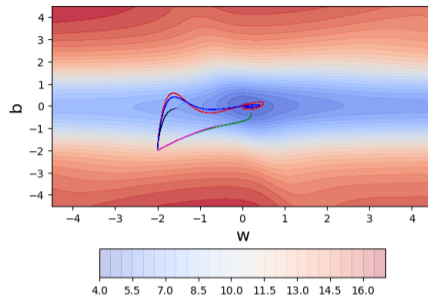
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, beta1 = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = beta1 * v_w + (1 - beta1) dw**2
        v_b = beta1 * v_b + (1 - beta1) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



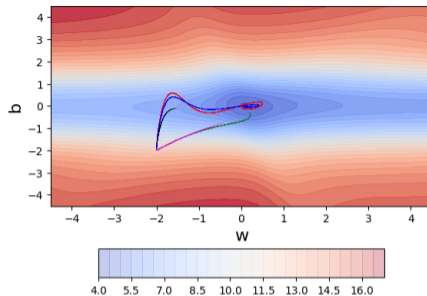
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



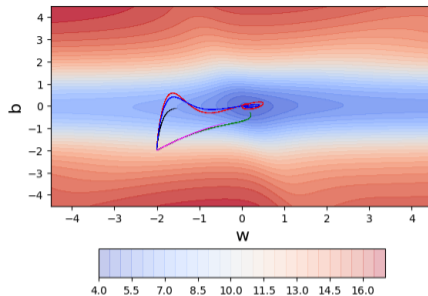
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



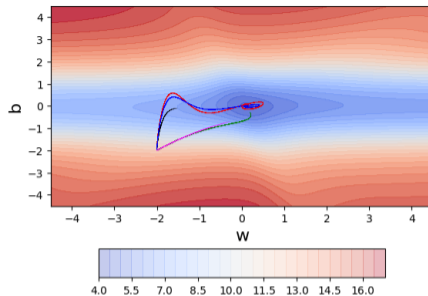
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



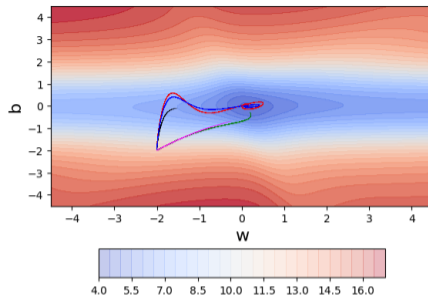
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



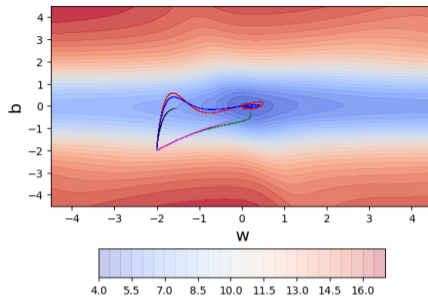
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



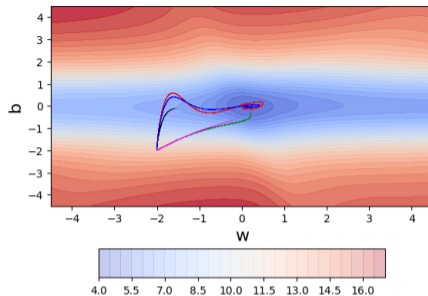
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



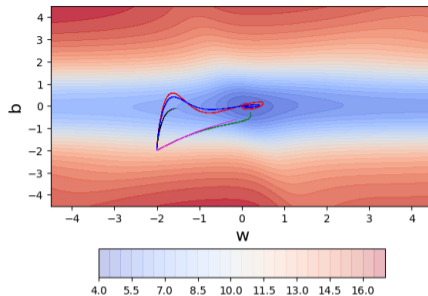
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



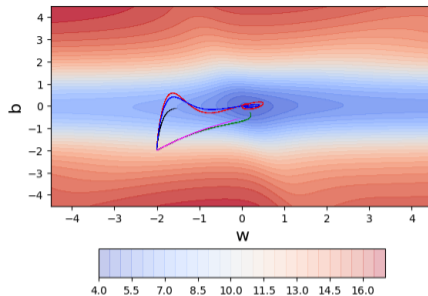
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



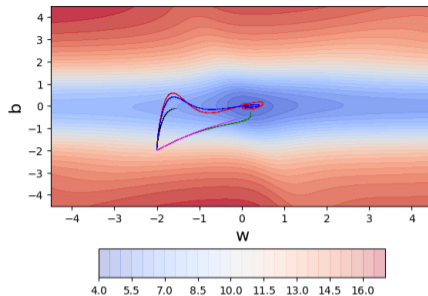
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

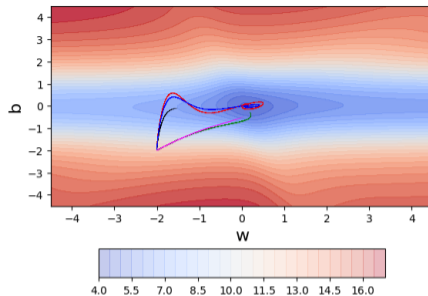
```



```
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



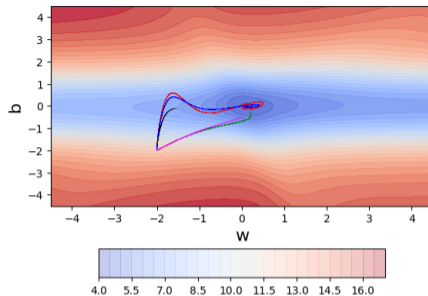
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



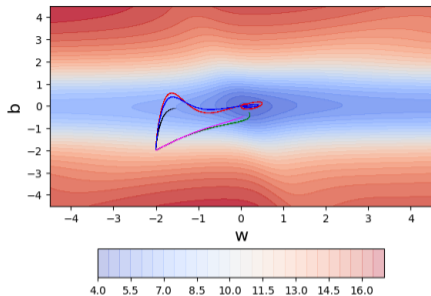
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



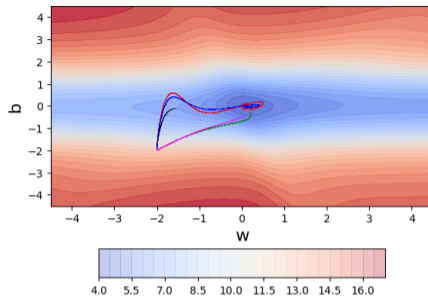
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



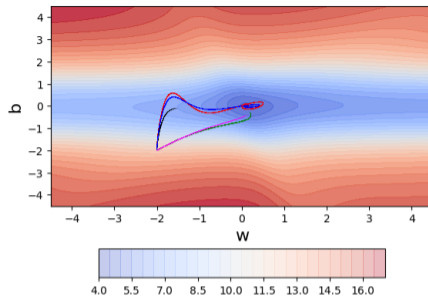
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



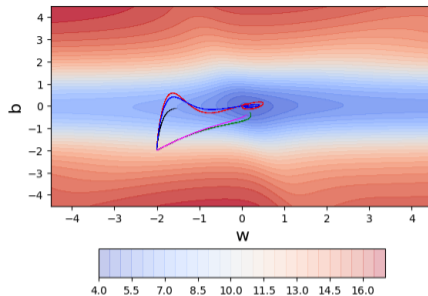
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



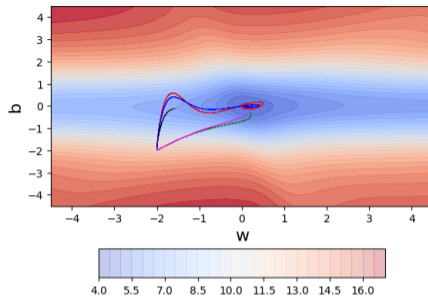
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



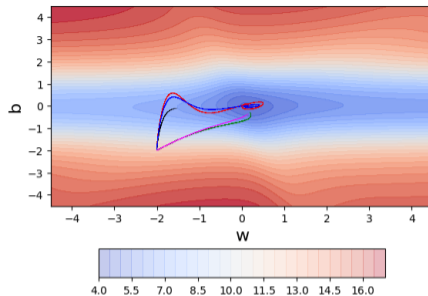
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



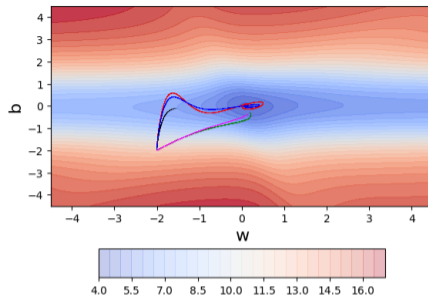
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



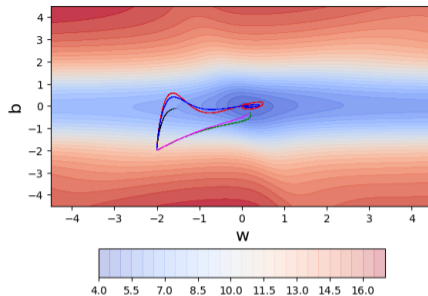
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



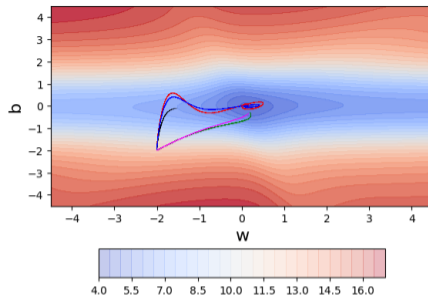
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



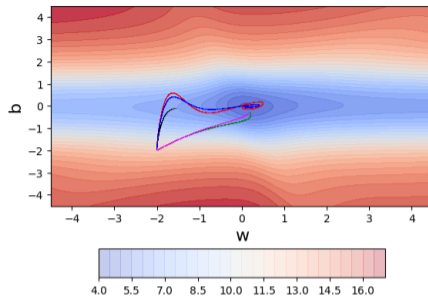
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



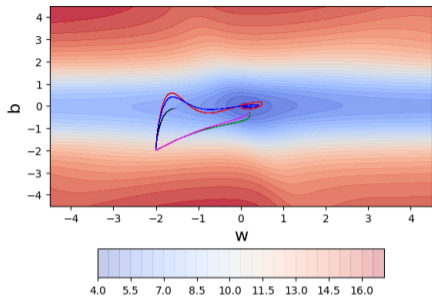
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



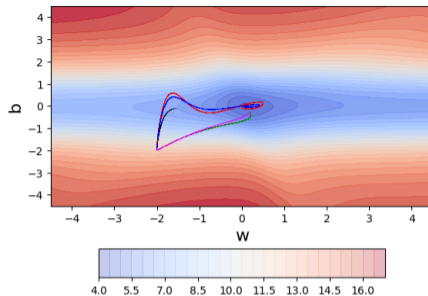
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



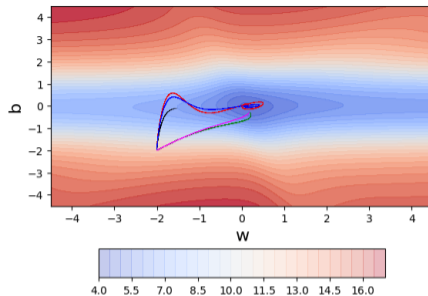
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

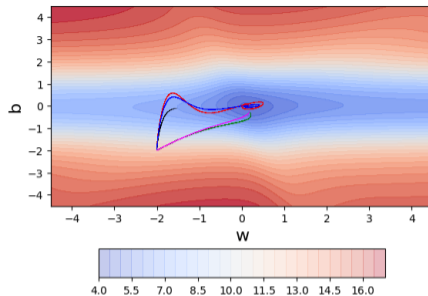
```



```
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



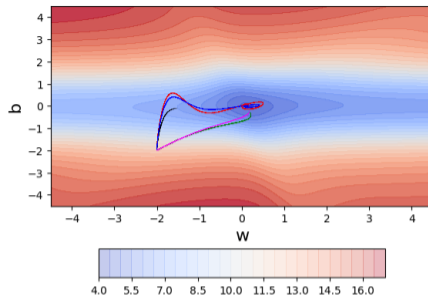
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



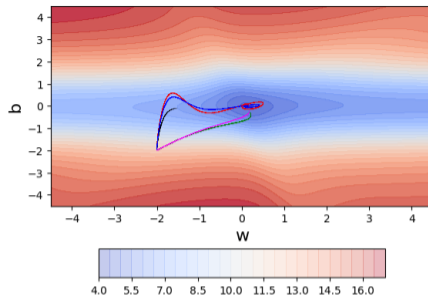
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



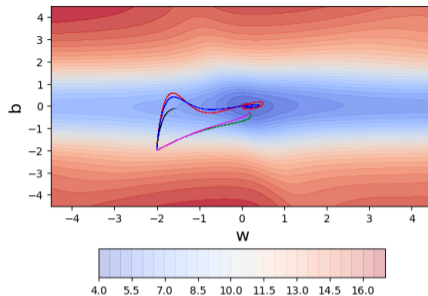
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



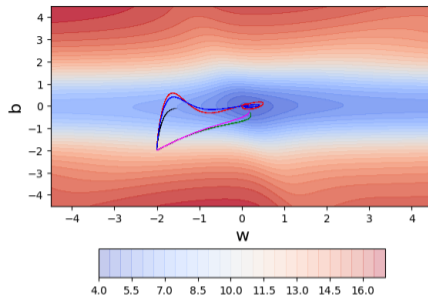
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



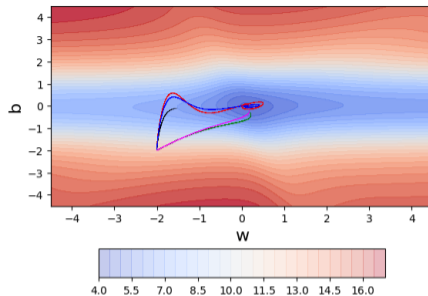
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

```



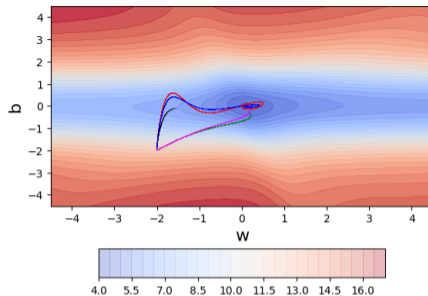
```

def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db

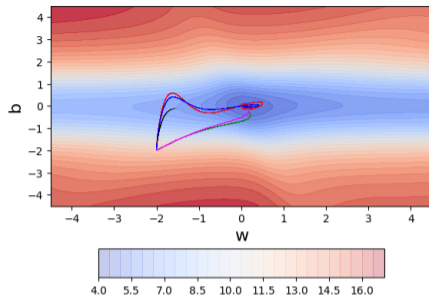
```



```
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```

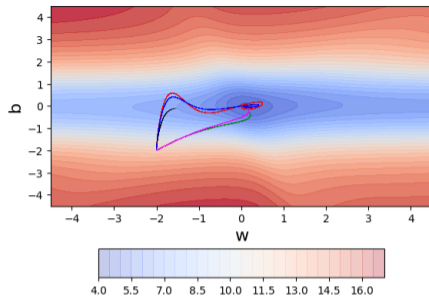


- Adagrad got stuck when it was close to convergence (it was no longer able to move in the vertical ( $b$ ) direction because of the decayed learning rate)

```
def do_rmsprop() :
    w, b, eta = init_w, init_b, 0.1
    v_w, b_updates, eps, betal = 0, 0, 1e-8, 0.9
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        v_w = betal * v_w + (1 - betal) dw**2
        v_b = betal * v_b + (1 - betal) db**2

        w = w - (eta / np.sqrt(v_w + eps)) * dw
        b = b - (eta / np.sqrt(v_b + eps)) * db
```



- Adagrad got stuck when it was close to convergence (it was no longer able to move in the vertical ( $b$ ) direction because of the decayed learning rate)

- RMSProp overcomes this problem by being less aggressive on the decay

## Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad

## Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients

## Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients

## Update rule for Adam

## Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients

## Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$

## Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients

## Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$
$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$

## Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients

## Update rule for Adam

$$\begin{aligned}m_t &= \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t \\v_t &= \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2 \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t}\end{aligned}$$

## Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients
- In practice,  $\beta_1 = 0.9$  and  $\beta_2 = 0.999$

## Update rule for Adam

$$\begin{aligned}m_t &= \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t \\v_t &= \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2 \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \quad \hat{v}_t = \frac{v_t}{1 - \beta_2^t}\end{aligned}$$

## Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients
- In practice,  $\beta_1 = 0.9$  and  $\beta_2 = 0.999$

## Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$

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

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} * \hat{m}_t$$

## Intuition

- Do everything that RMSProp does to solve the decay problem of Adagrad
- Plus use a cumulative history of the gradients
- In practice,  $\beta_1 = 0.9$  and  $\beta_2 = 0.999$

## Update rule for Adam

$$m_t = \beta_1 * m_{t-1} + (1 - \beta_1) * \nabla w_t$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2) * (\nabla w_t)^2$$

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

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} * \hat{m}_t$$

... and a similar set of equations for  $b_t$

```

def do_adam() :
    w_b dw_db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [], []

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0,
        0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

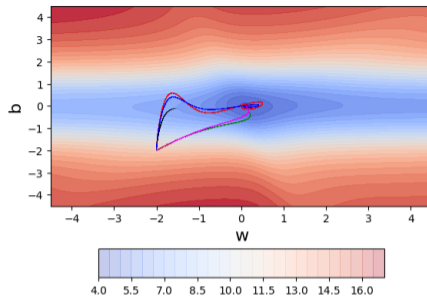
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b dw_db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

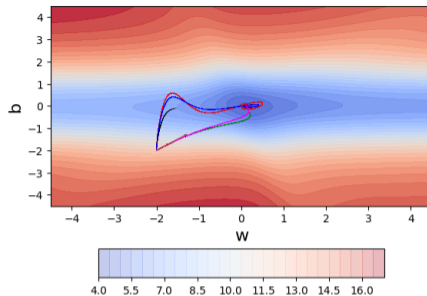
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b_dw_db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [], []

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0,
        0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

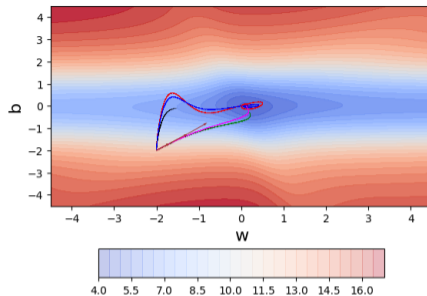
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b_dw_db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [], []

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0,
        0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

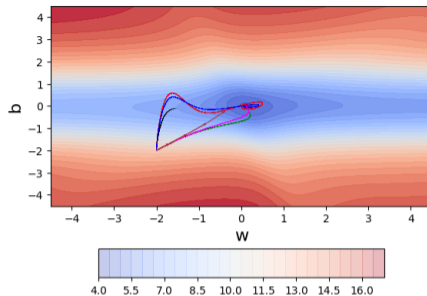
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

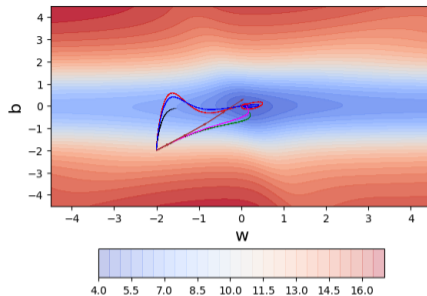
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

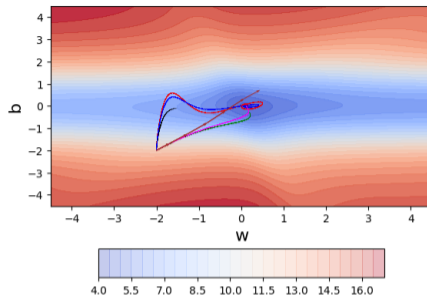
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

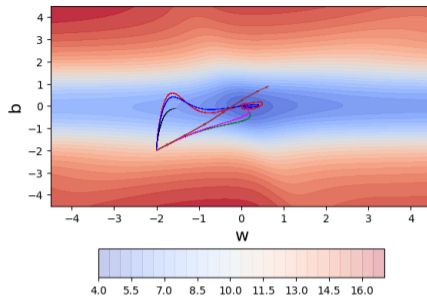
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

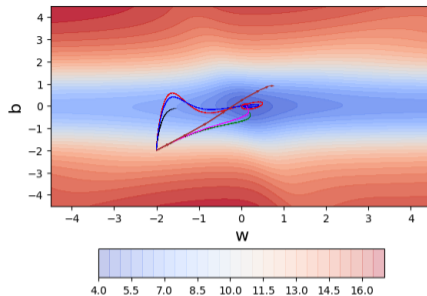
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

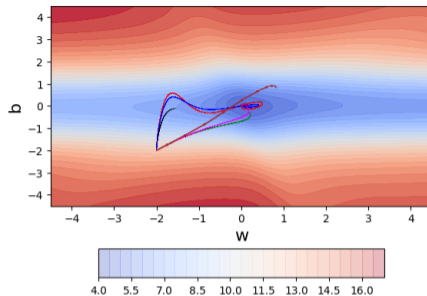
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b dw_db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

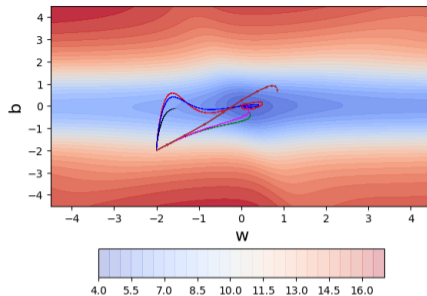
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

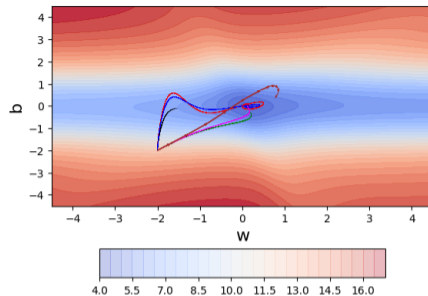
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

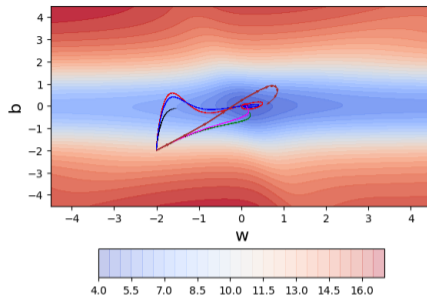
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

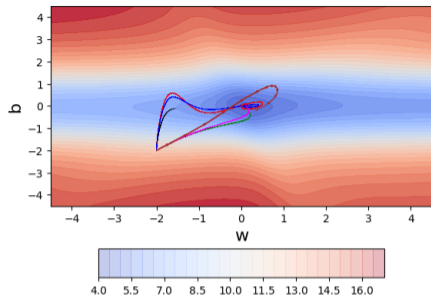
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

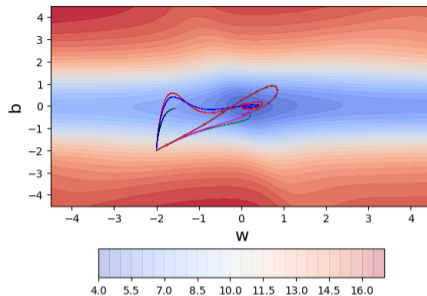
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

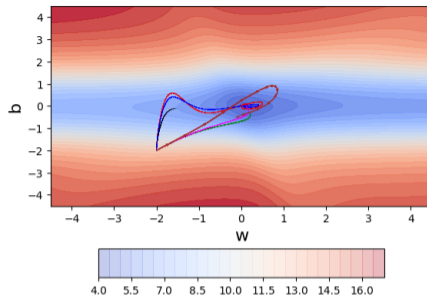
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b dw_db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

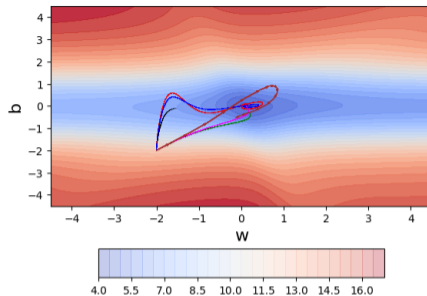
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

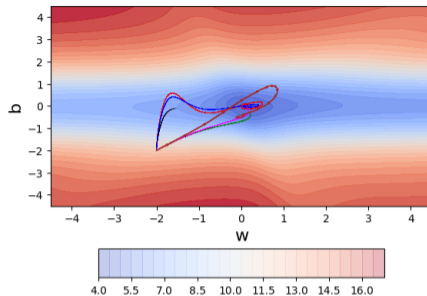
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

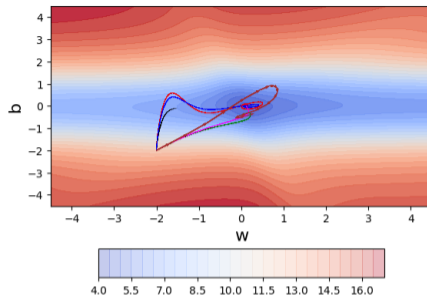
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b dw_db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

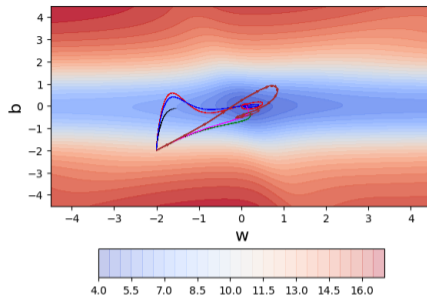
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

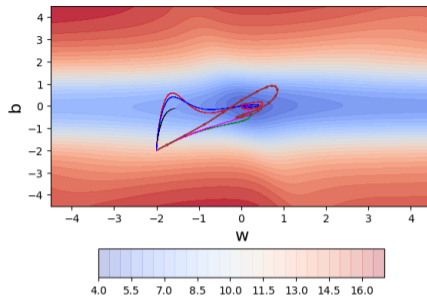
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

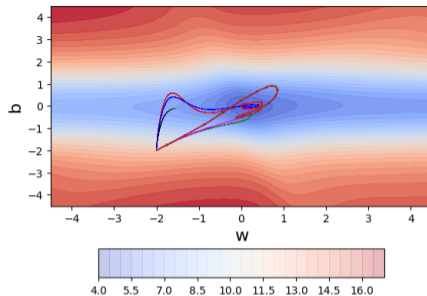
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

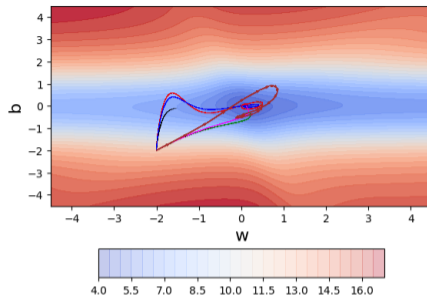
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

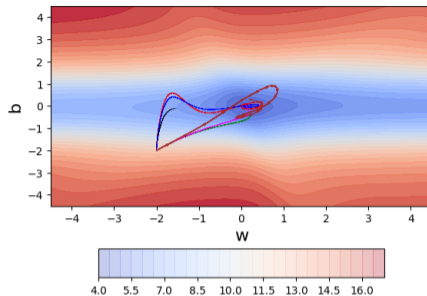
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

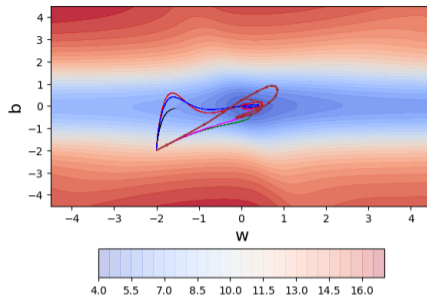
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

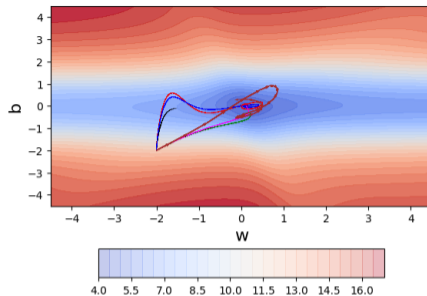
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

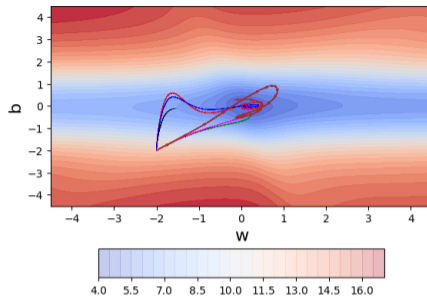
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

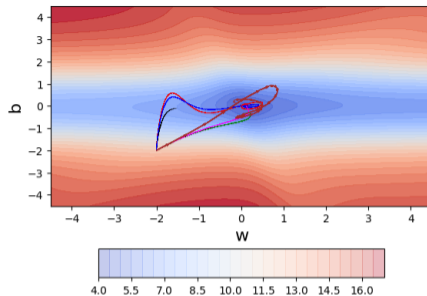
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b dw_db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

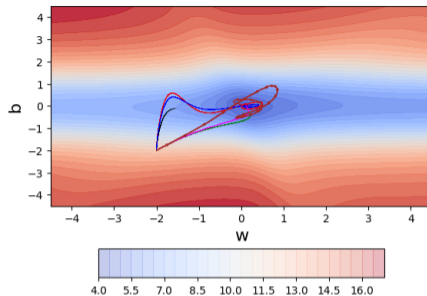
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

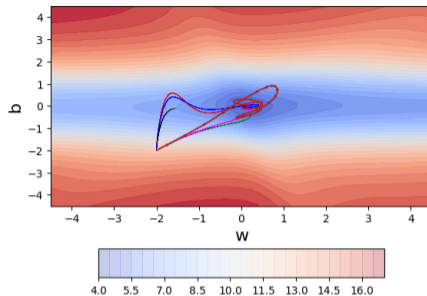
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

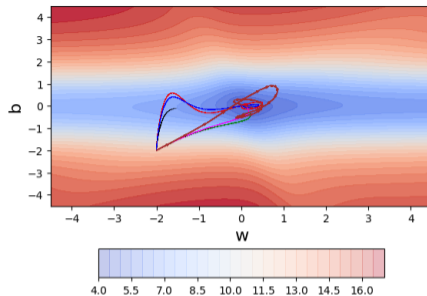
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b dw_db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

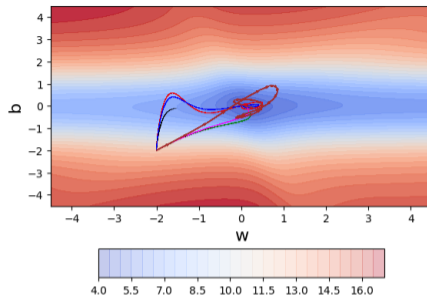
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

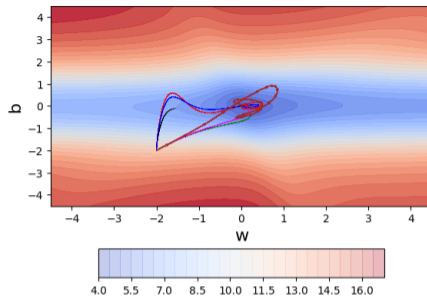
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

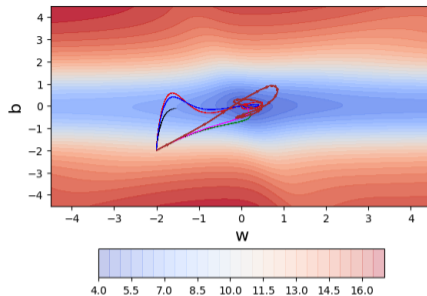
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b,dw_db = [(init_w, init_b,0,0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

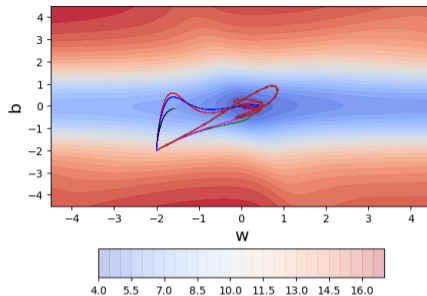
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



```

def do_adam() :
    w_b, dw_db = [(init_w, init_b, 0, 0)]
    w_history, b_history, error_history = [], [], [
    ]

    w, b, eta, mini_batch_size, num_points_seen =
        init_w, init_b, 0.1, 10, 0
    m_w, m_b, v_w, v_b, eps, beta1, beta2 = 0, 0, 0
        , 0, 1e-8, 0.9, 0.999
    for i in range(max_epochs) :
        dw, db = 0, 0
        for x,y in zip(X, Y) :
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)

        m_w = beta1 * m_w + (1-beta1)*dw
        m_b = beta1 * m_b + (1-beta1)*db

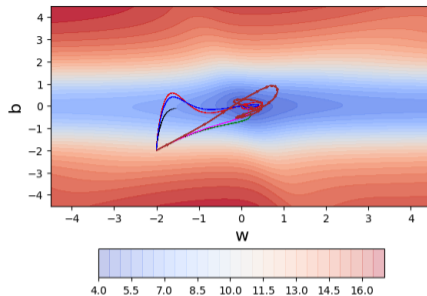
        v_w = beta2 * v_w + (1-beta2)*dw**2
        v_b = beta2 * v_b + (1-beta2)*db**2

        m_w = m_w/(1-math.pow(beta1,i+1))
        m_b = m_b/(1-math.pow(beta1,i+1))

        v_w = v_w/(1-math.pow(beta2,i+1))
        v_b = v_b/(1-math.pow(beta2,i+1))

        w = w - (eta / np.sqrt(v_w + eps)) * m_w
        b = b - (eta / np.sqrt(v_b + eps)) * m_b

```



- As expected, taking a cumulative history gives a speed up ...

## Million dollar question: Which algorithm to use in practice

- Adam seems to be more or less the default choice now ( $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 1e - 8$  )

## Million dollar question: Which algorithm to use in practice

- Adam seems to be more or less the default choice now ( $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 1e - 8$  )
- Although it is supposed to be robust to initial learning rates, we have observed that for sequence generation problems  $\eta = 0.001, 0.0001$  works best

## Million dollar question: Which algorithm to use in practice

- Adam seems to be more or less the default choice now ( $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 1e - 8$  )
- Although it is supposed to be robust to initial learning rates, we have observed that for sequence generation problems  $\eta = 0.001, 0.0001$  works best
- Having said that, many papers report that SGD with momentum (Nesterov or classical) with a simple annealing learning rate schedule also works well in practice (typically, starting with  $\eta = 0.001, 0.0001$  for sequence generation problems)

## Million dollar question: Which algorithm to use in practice

- Adam seems to be more or less the default choice now ( $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 1e - 8$  )
- Although it is supposed to be robust to initial learning rates, we have observed that for sequence generation problems  $\eta = 0.001, 0.0001$  works best
- Having said that, many papers report that SGD with momentum (Nesterov or classical) with a simple annealing learning rate schedule also works well in practice (typically, starting with  $\eta = 0.001, 0.0001$  for sequence generation problems)
- Adam might just be the best choice overall!!

## Million dollar question: Which algorithm to use in practice

- Adam seems to be more or less the default choice now ( $\beta_1 = 0.9$ ,  $\beta_2 = 0.999$  and  $\epsilon = 1e - 8$  )
- Although it is supposed to be robust to initial learning rates, we have observed that for sequence generation problems  $\eta = 0.001, 0.0001$  works best
- Having said that, many papers report that SGD with momentum (Nesterov or classical) with a simple annealing learning rate schedule also works well in practice (typically, starting with  $\eta = 0.001, 0.0001$  for sequence generation problems)
- Adam might just be the best choice overall!!
- Some recent work suggest that there is a problem with Adam and it will not converge in some cases



