



第五讲：梯度下降算法及其变体

张盛平

s.zhang@hit.edu.cn

计算学部
哈尔滨工业大学

2021 年秋季学期



致谢

- 讲稿中很多资料或素材来源于网络，包括国外一些大学的相关课程、一些博客、维基百科等。后面不一一列举来源，在此一并表示感谢
- 引用网络资源时，由于本人的理解能力，可能存在一些偏差
- 本讲稿会经常更新



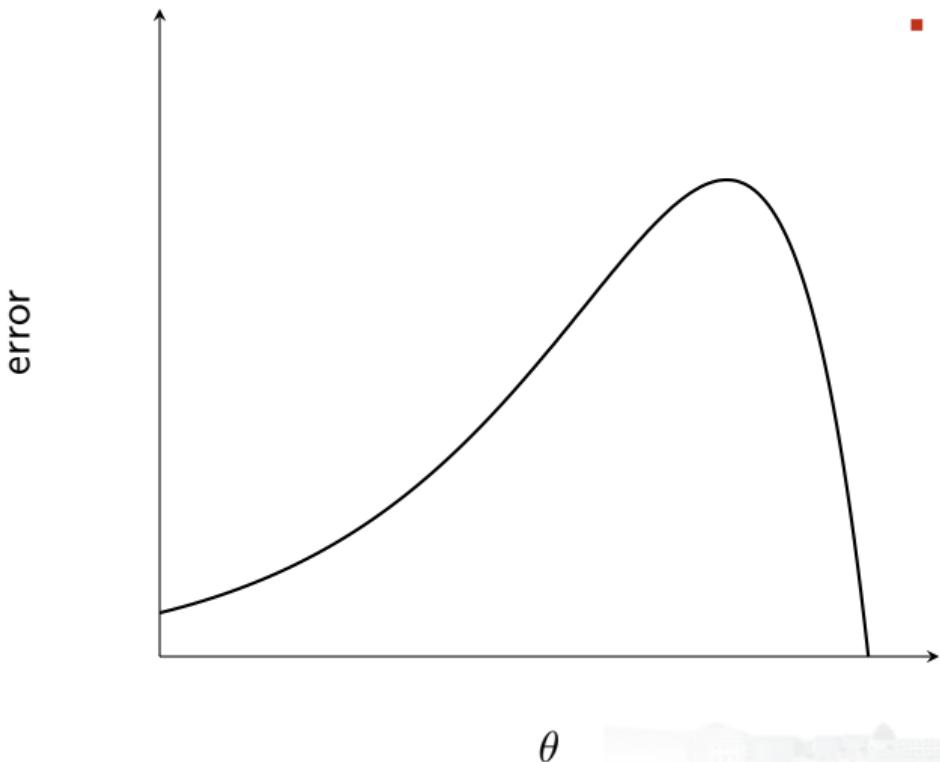
- 3D 坐标系对损失函数曲面的遍历进行可视化不太方便



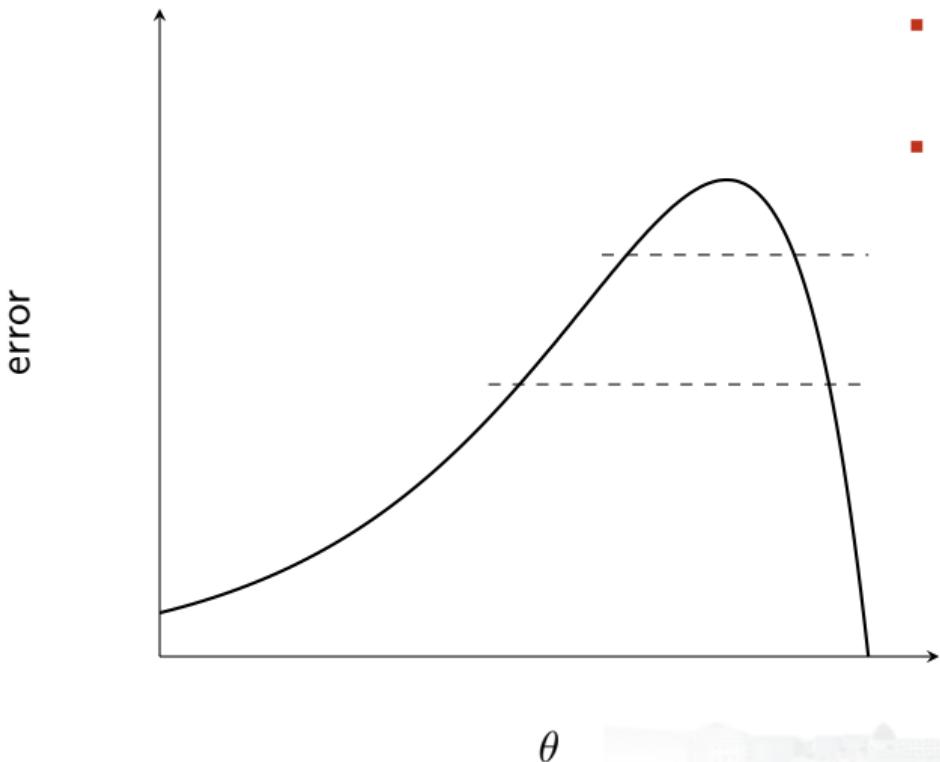
- 3D 坐标系对损失函数曲面的遍历进行可视化不太方便
- 能否借助 2D 可视化来遍历损失函数曲面 ?



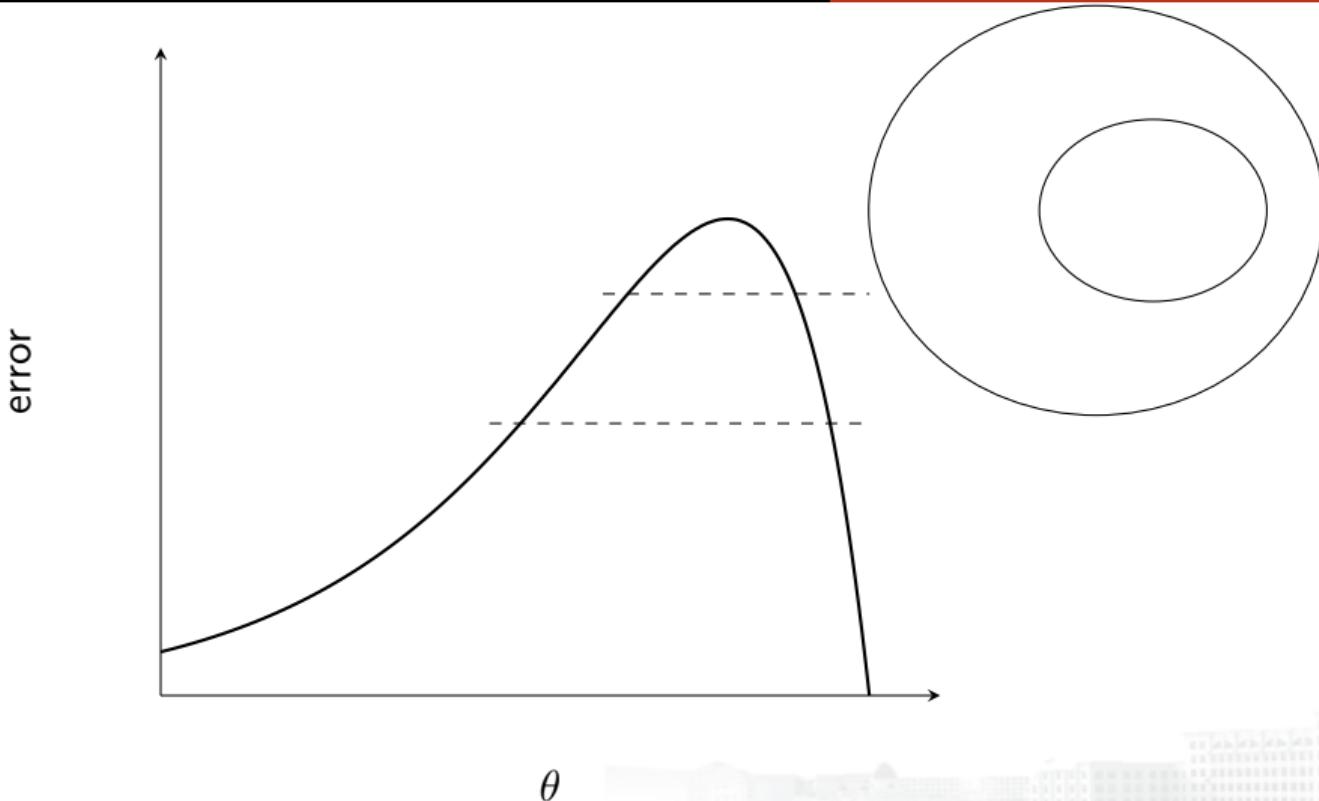
- 3D 坐标系对损失函数曲面的遍历进行可视化不太方便
- 能否借助 2D 可视化来遍历损失函数曲面 ?
- 等高线图 (*contour maps*)



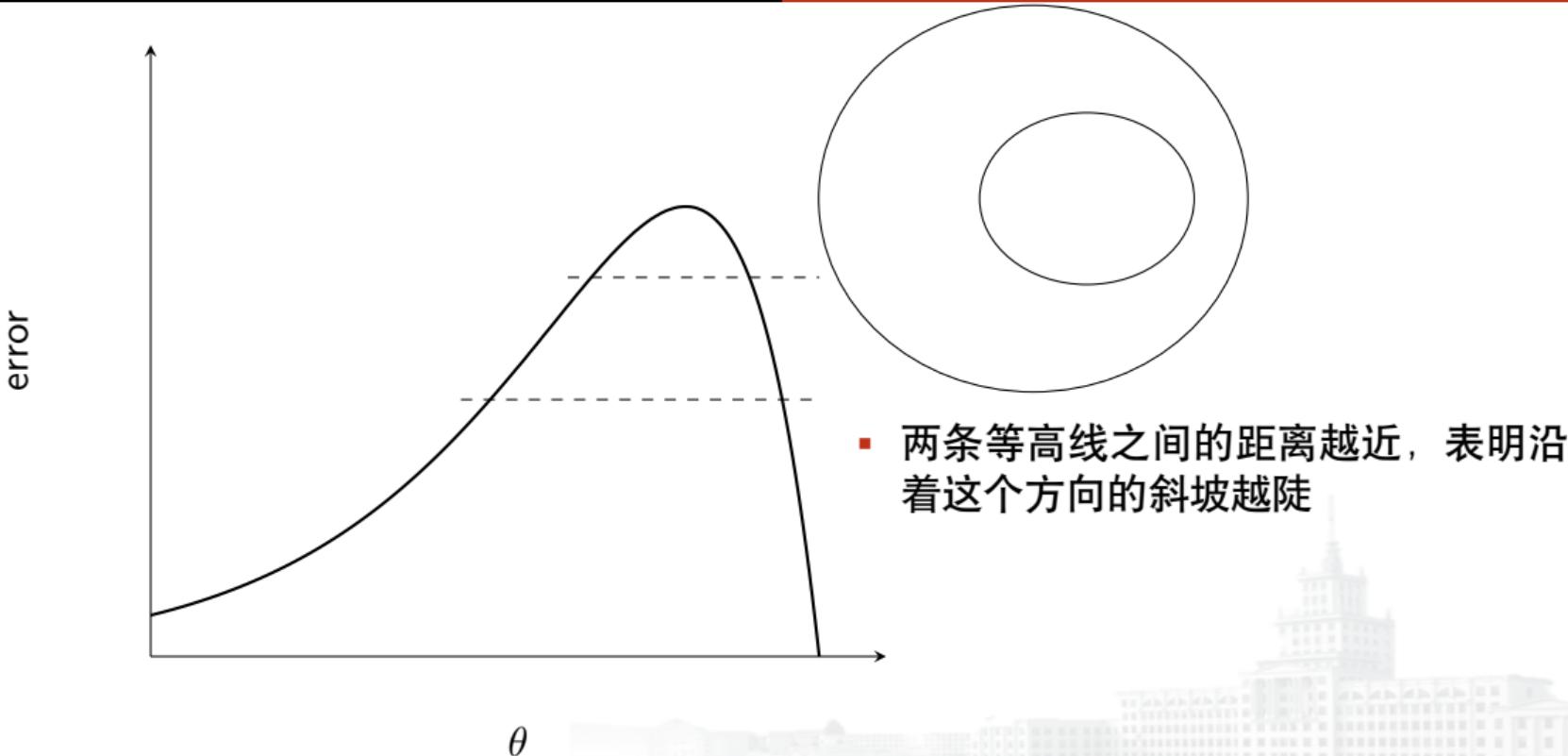
- 假设沿着垂直竖轴将误差曲面等间隔的切片

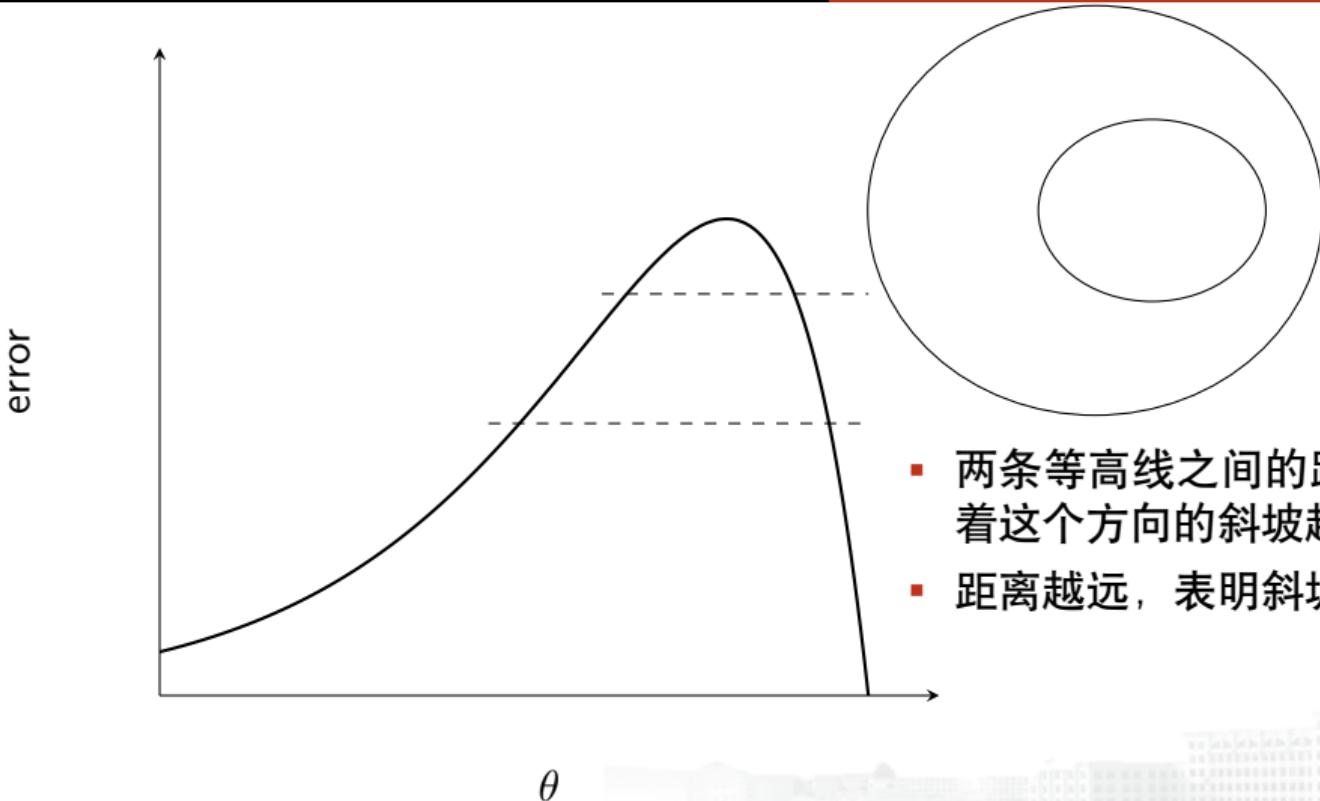


- 假设沿着垂直竖轴将误差曲面等间隔的切片
- 从上往下俯视看起来像什么？



θ

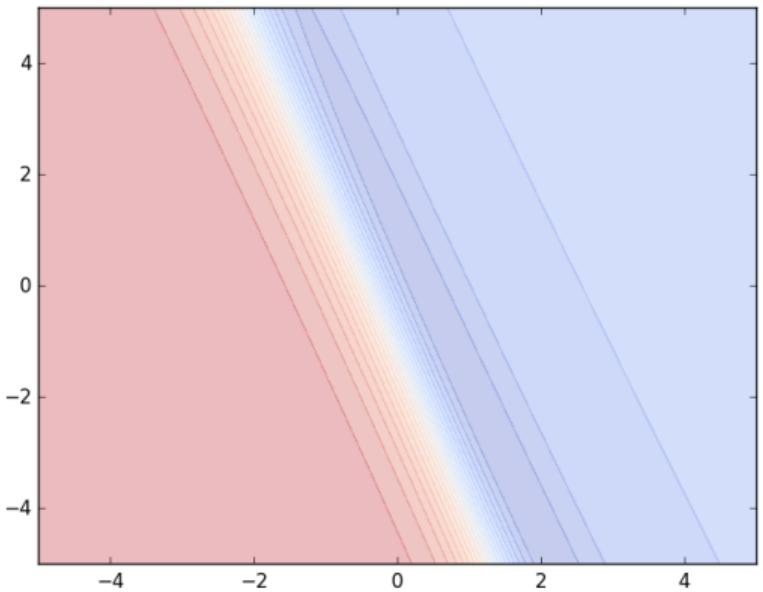




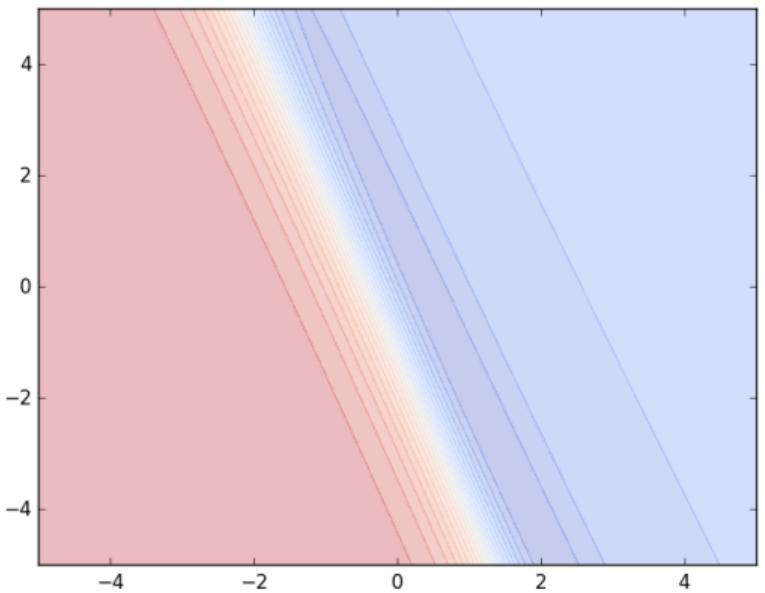
- 两条等高线之间的距离越近，表明沿着这个方向的斜坡越陡
- 距离越远，表明斜坡越平坦



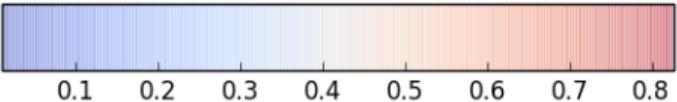
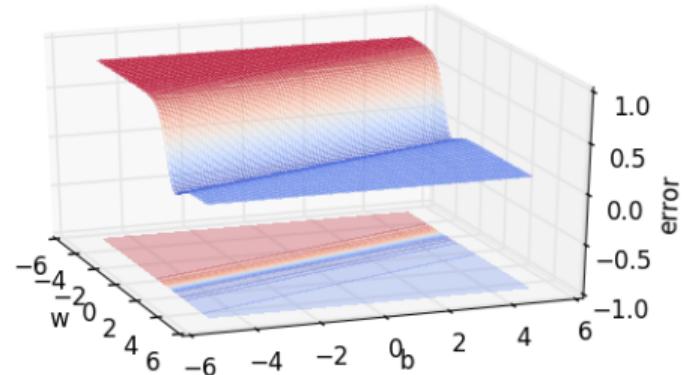
- 更多 3D 损失函数曲面和等高线的例子...

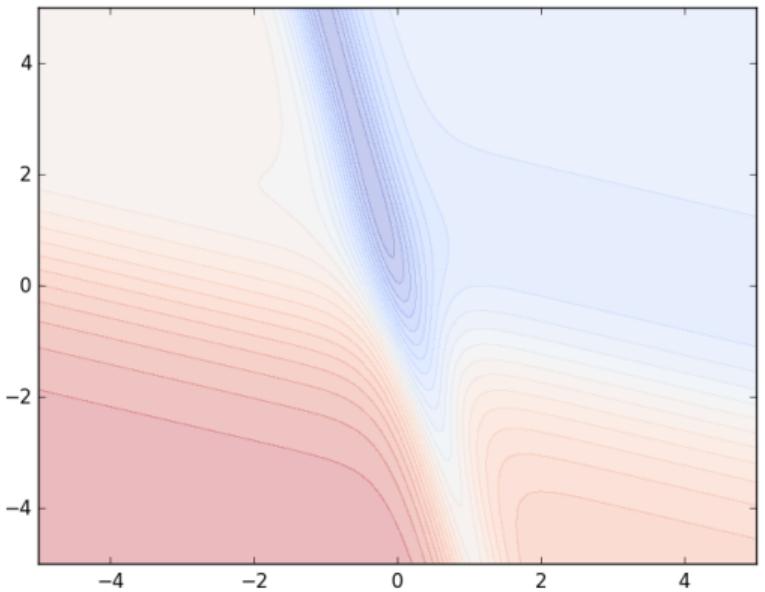


从等高线图推测 3D 损失函数曲面

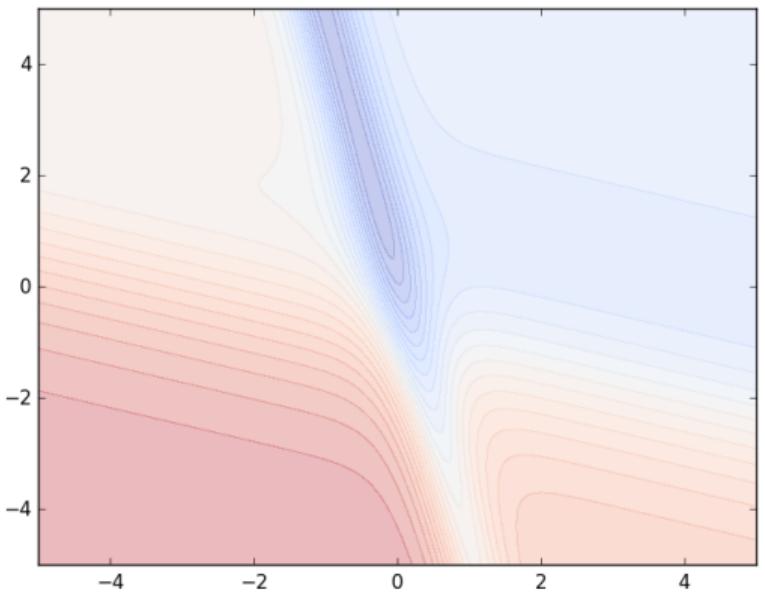


从等高线图推测 3D 损失函数曲面

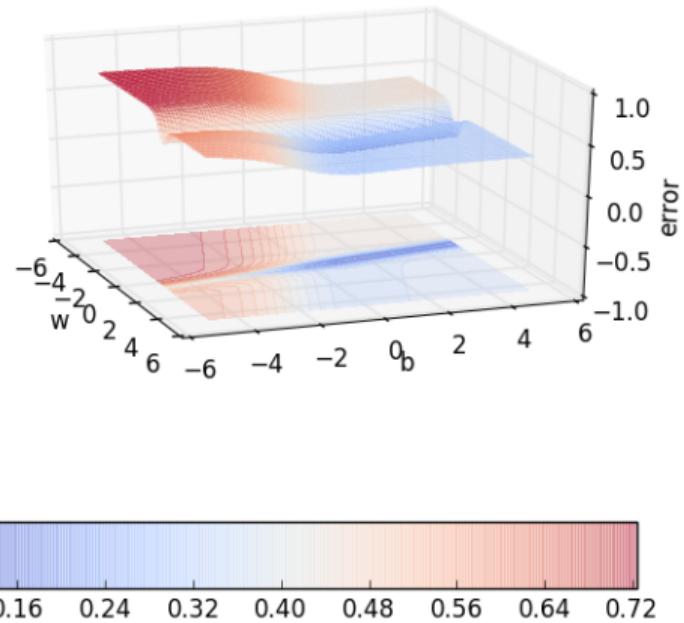


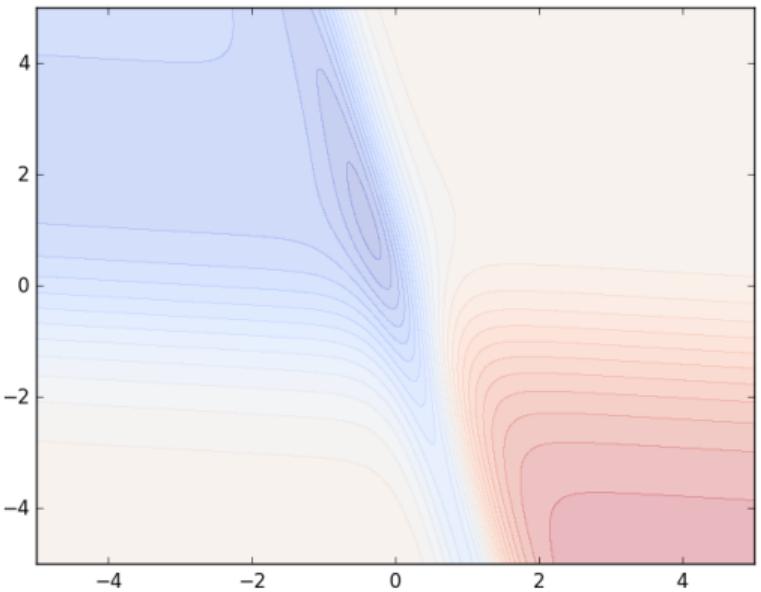


从等高线图推测 3D 损失函数曲面

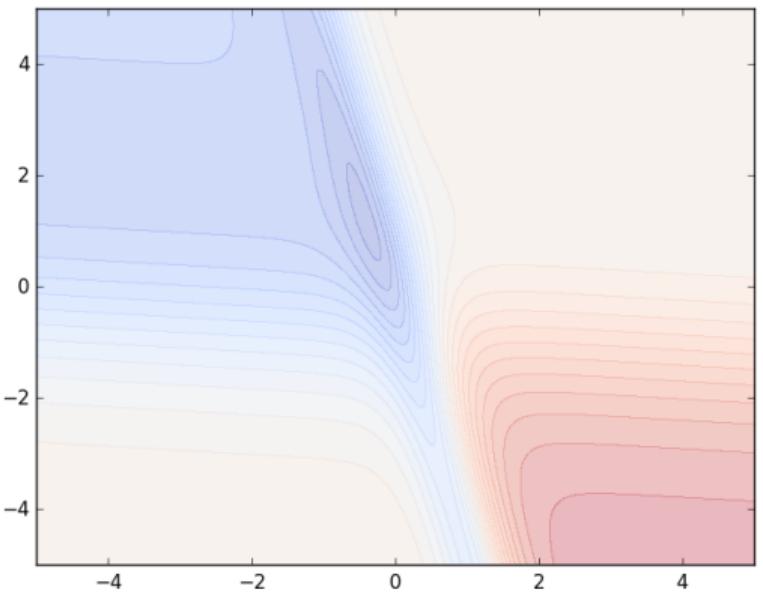


从等高线图推测 3D 损失函数曲面

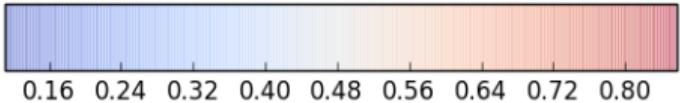
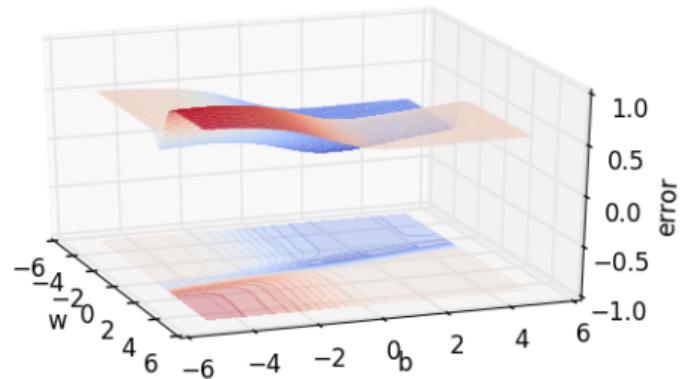




从等高线图推测 3D 损失函数曲面

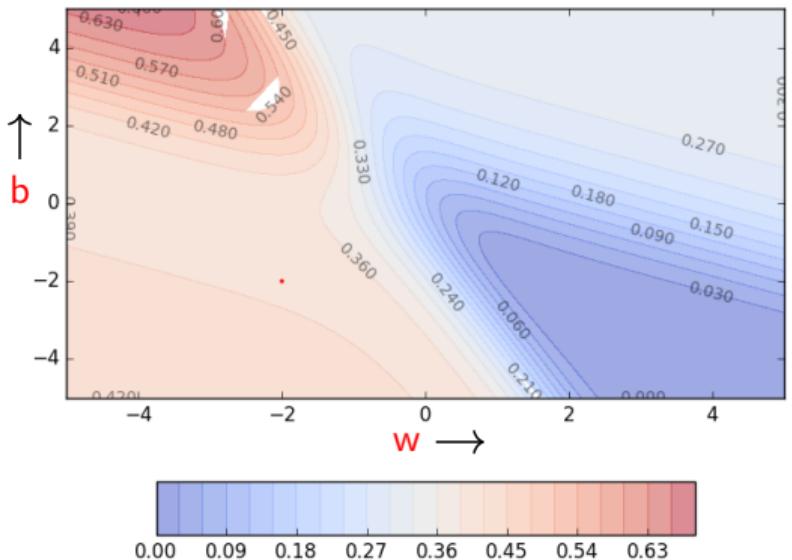


从等高线图推测 3D 损失函数曲面

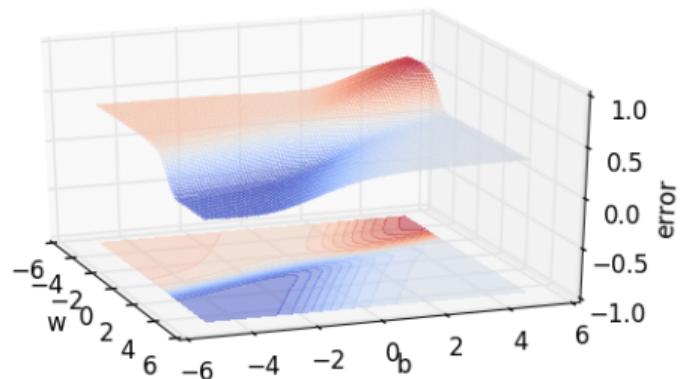


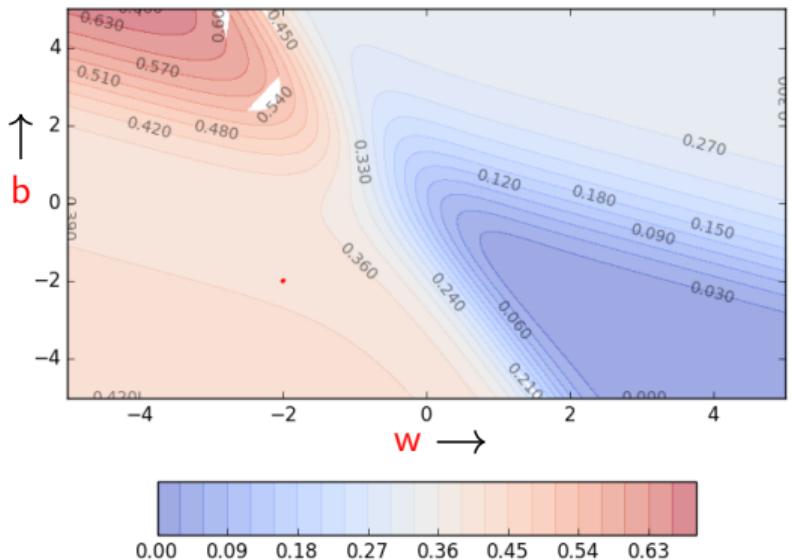


- 从等高线的视角，对梯度下降进行可视化

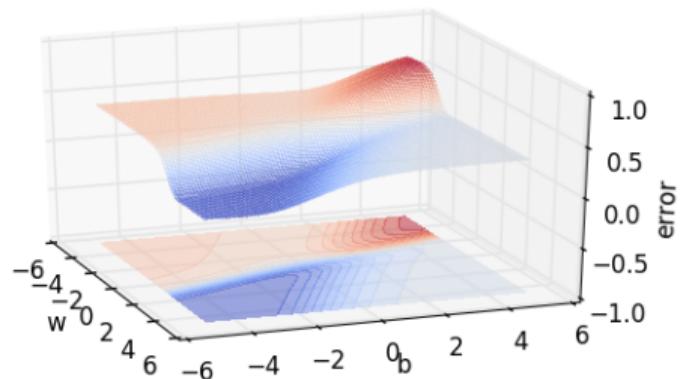


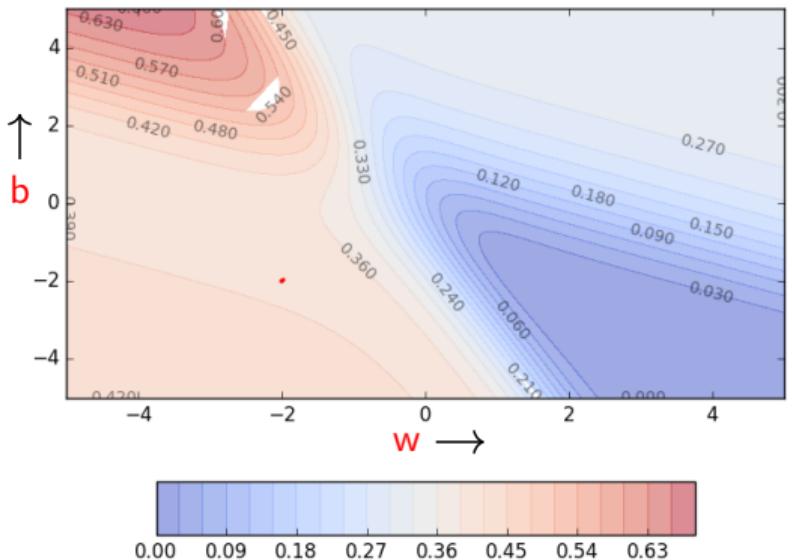
Gradient descent on the error surface



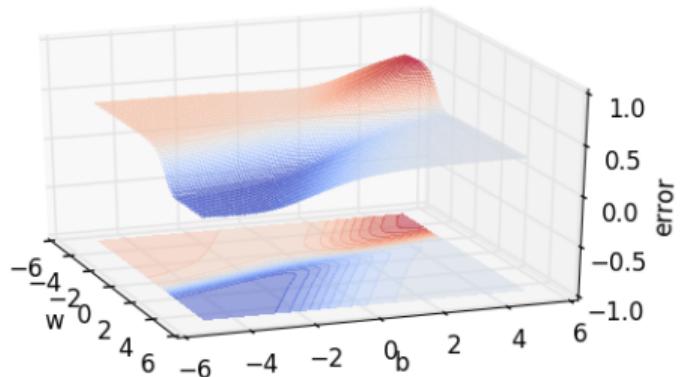


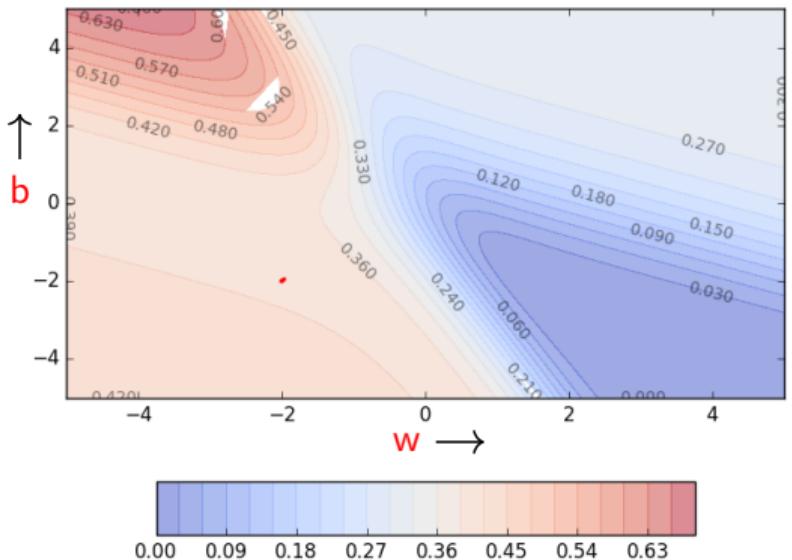
Gradient descent on the error surface



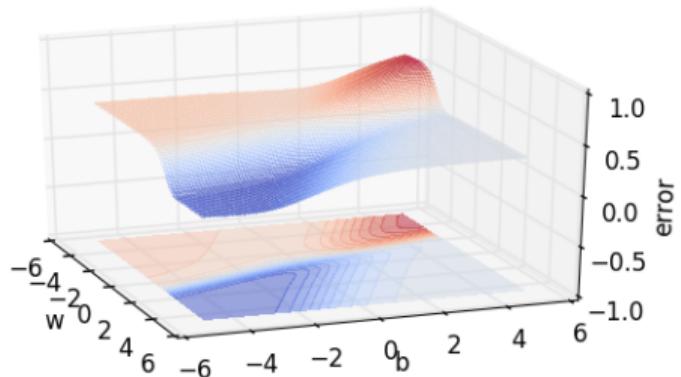


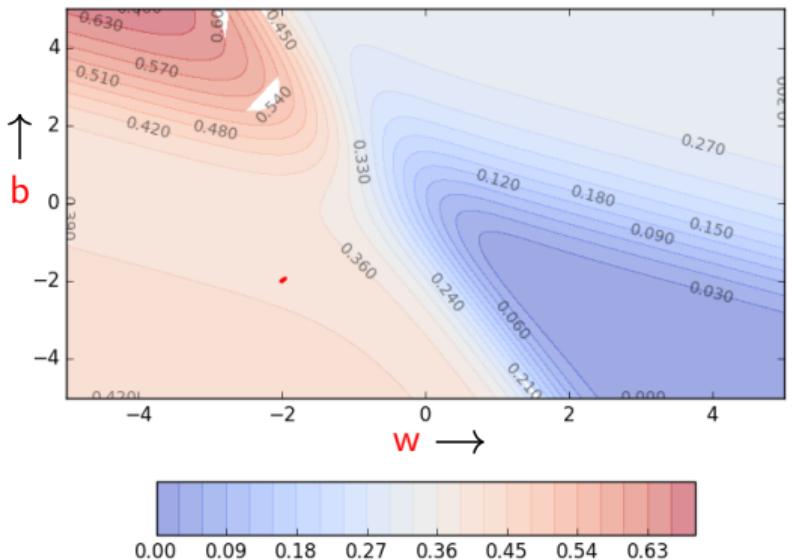
Gradient descent on the error surface



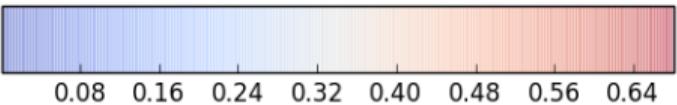
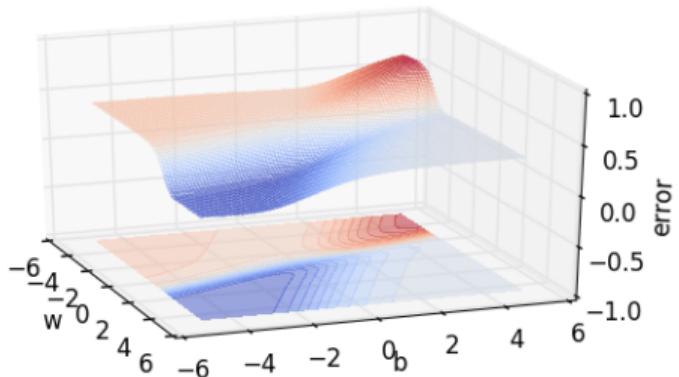


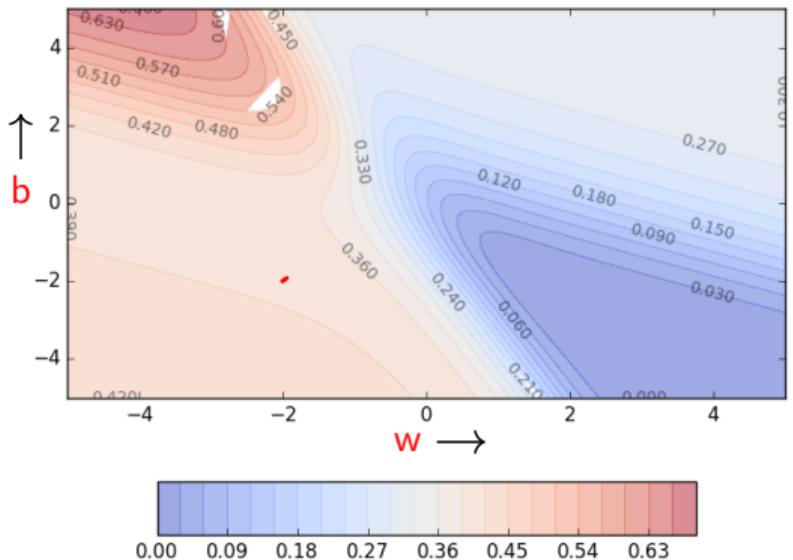
Gradient descent on the error surface



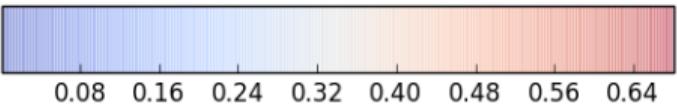
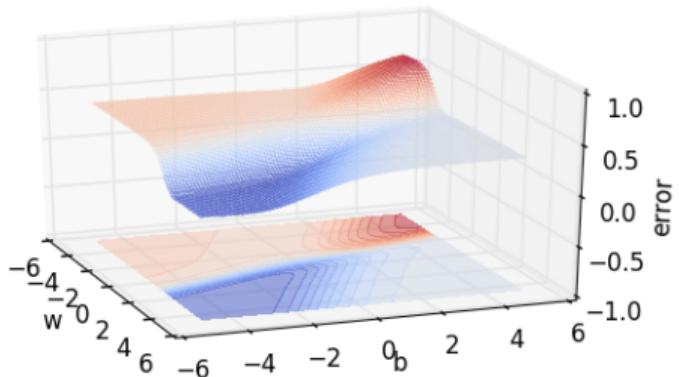


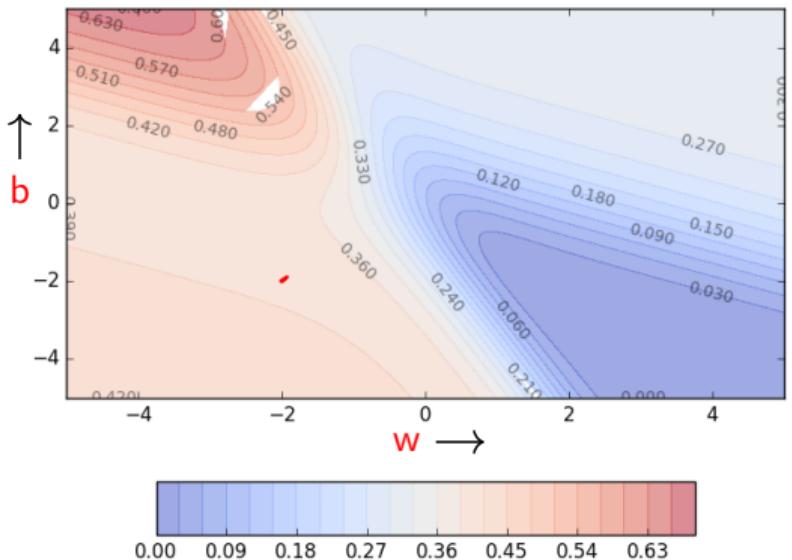
Gradient descent on the error surface



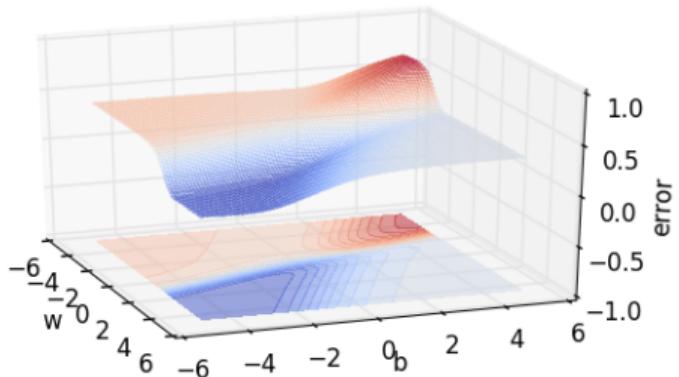


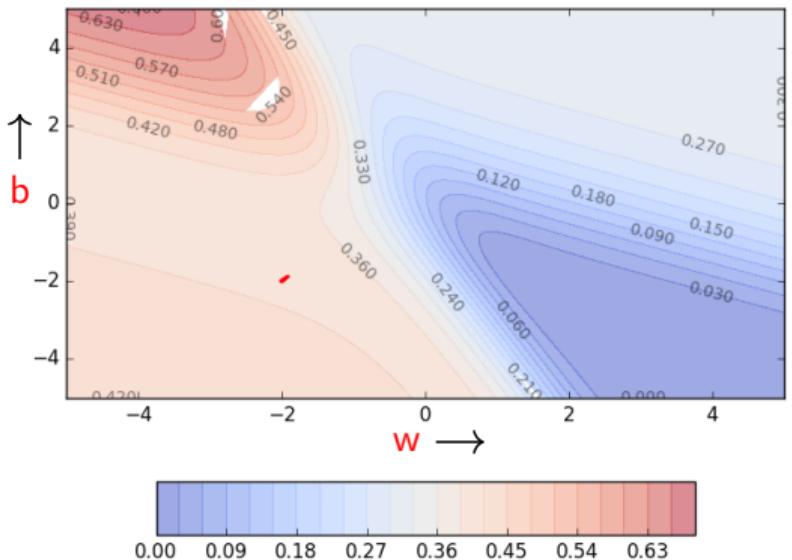
Gradient descent on the error surface



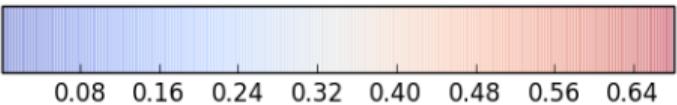
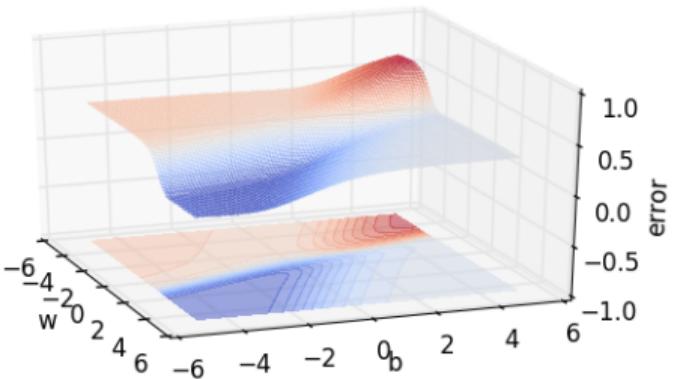


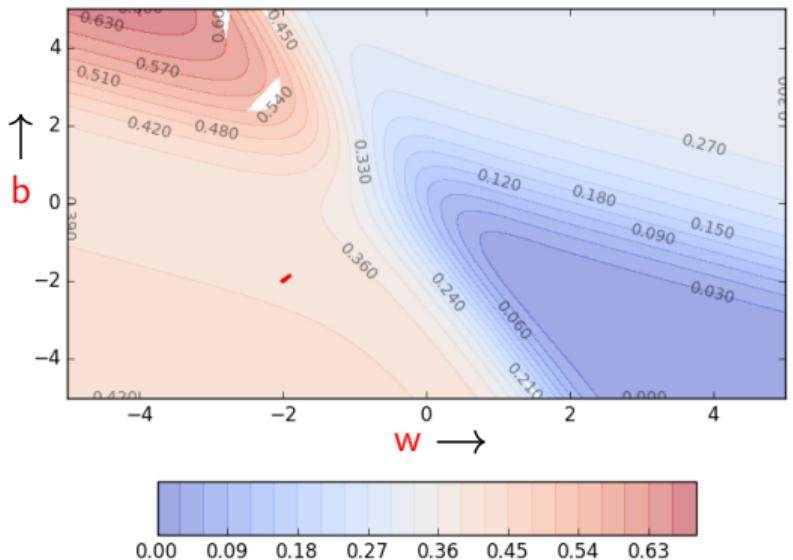
Gradient descent on the error surface



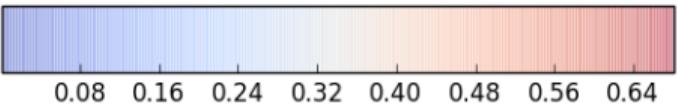
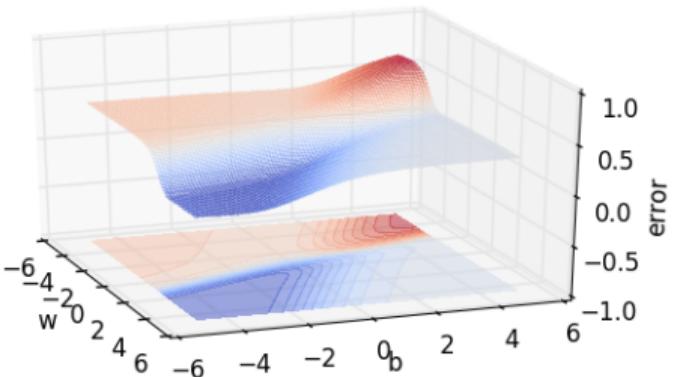


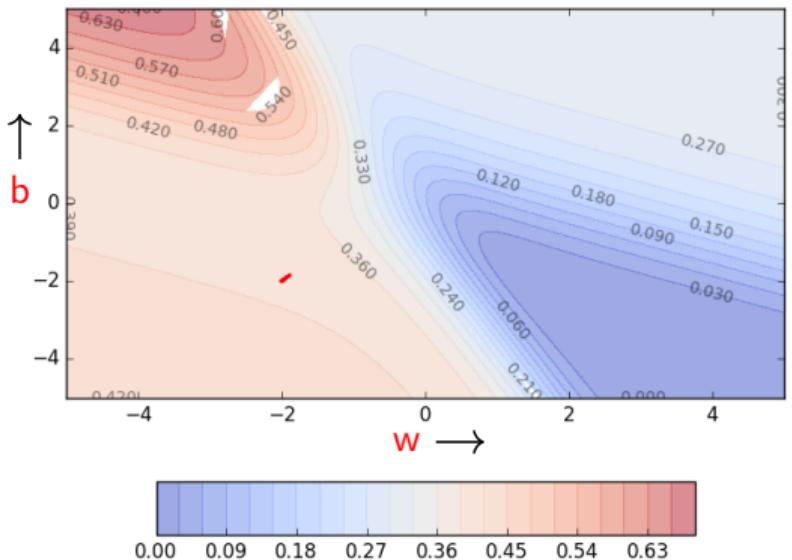
Gradient descent on the error surface



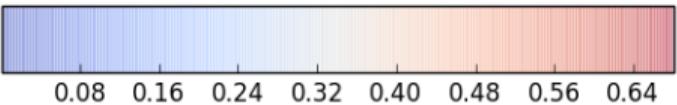
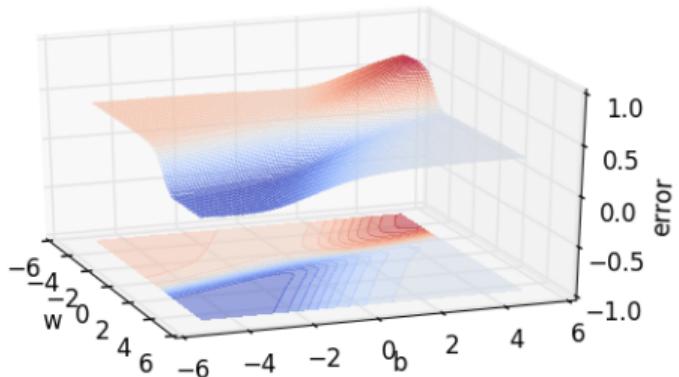


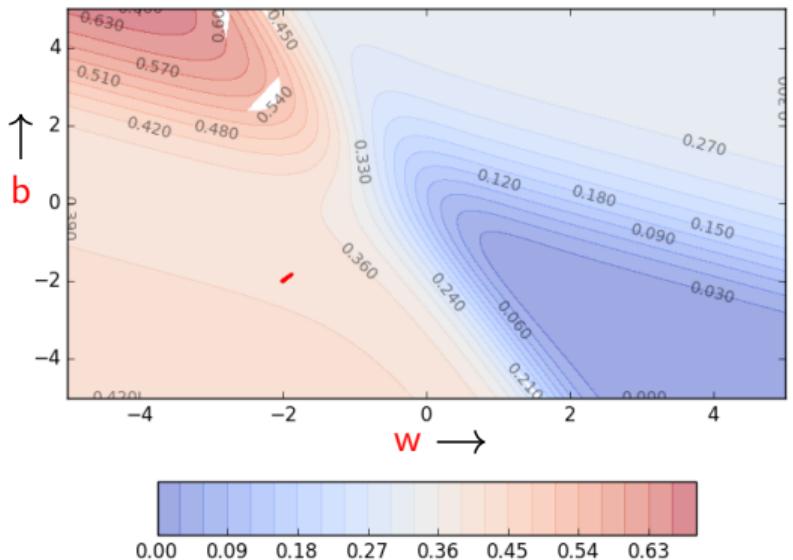
Gradient descent on the error surface



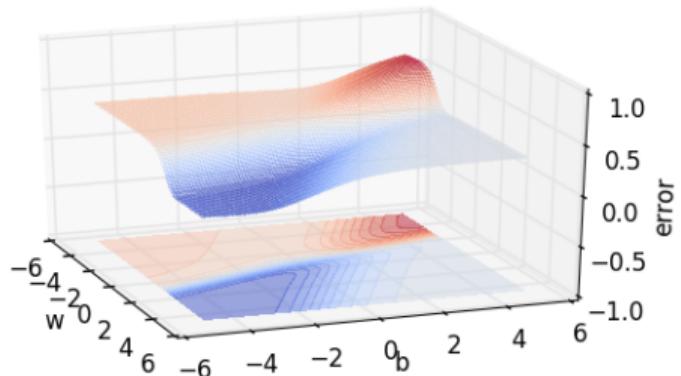


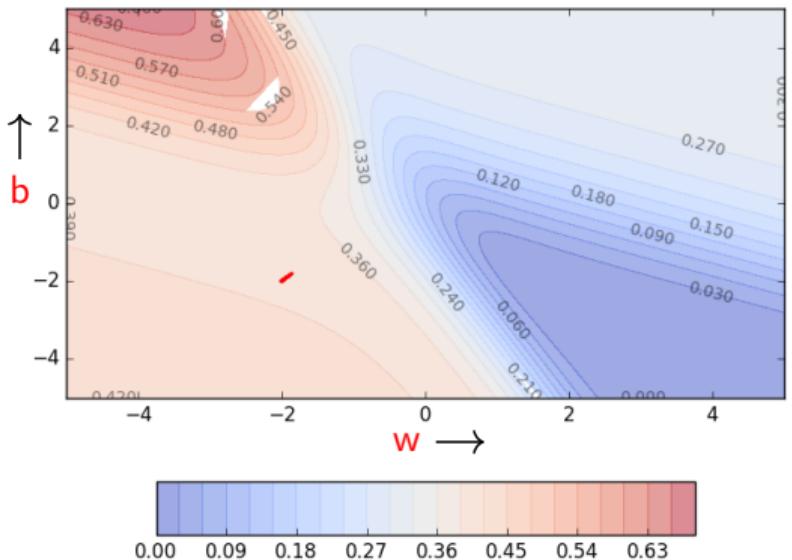
Gradient descent on the error surface



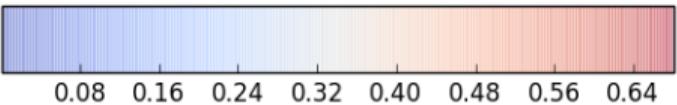
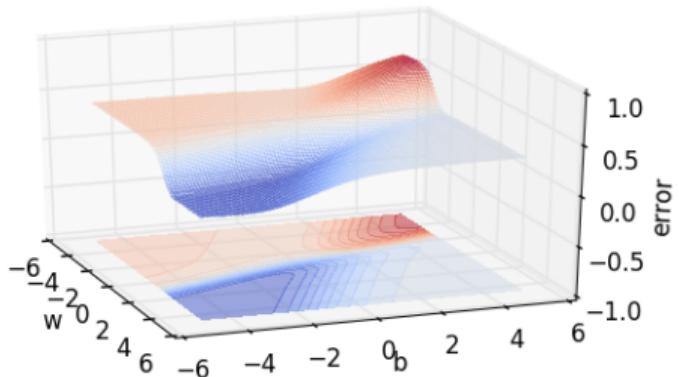


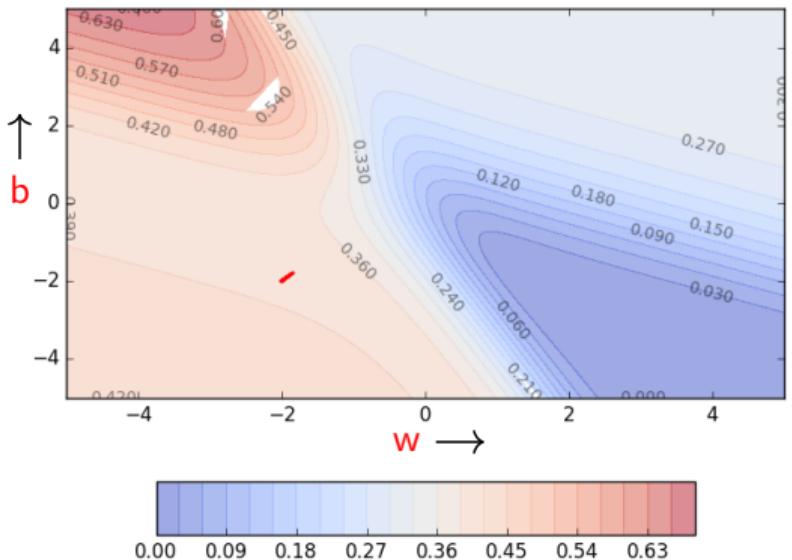
Gradient descent on the error surface



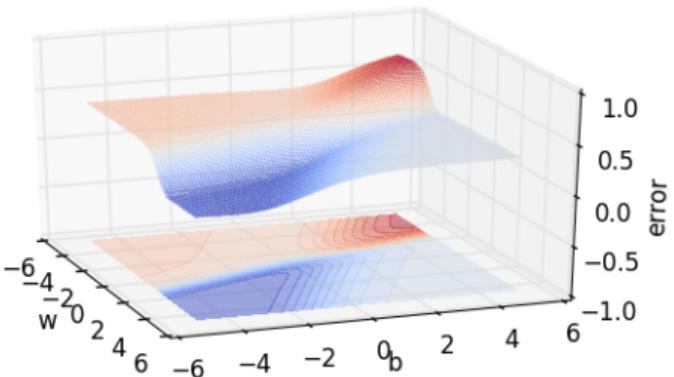


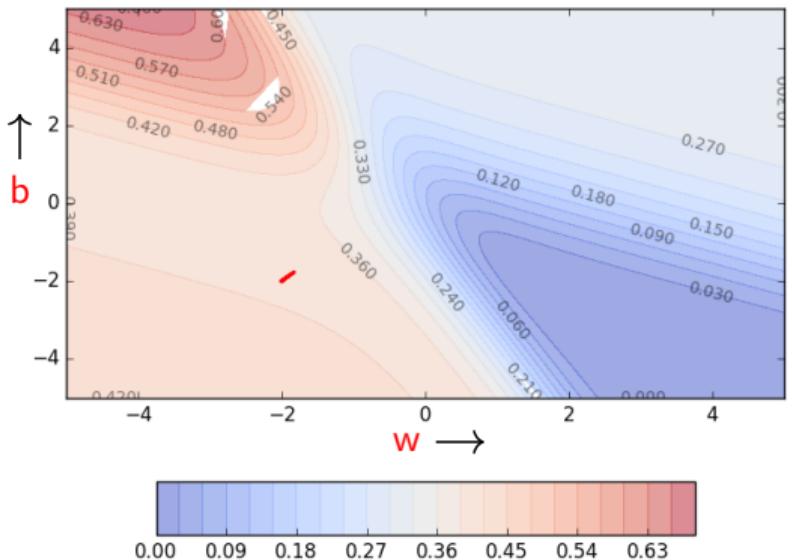
Gradient descent on the error surface



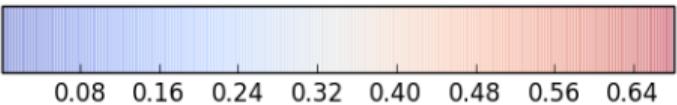
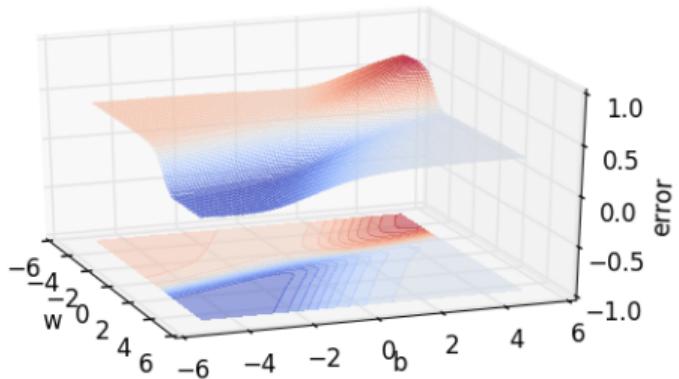


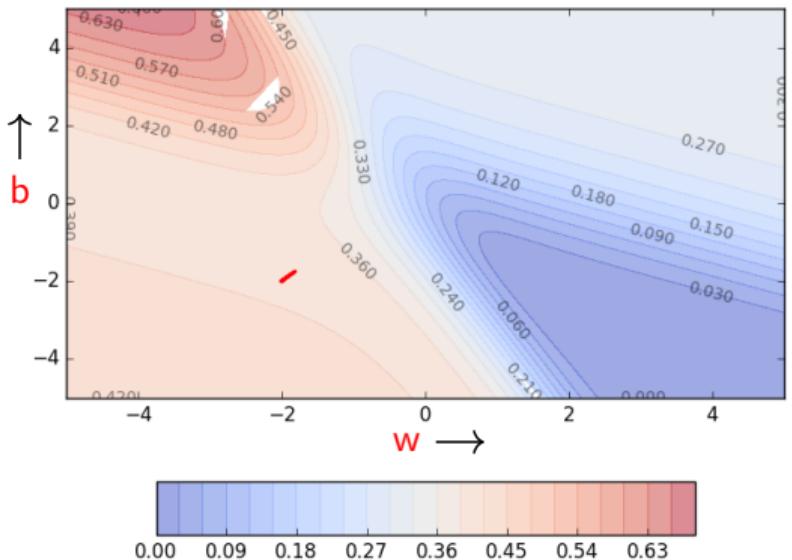
Gradient descent on the error surface



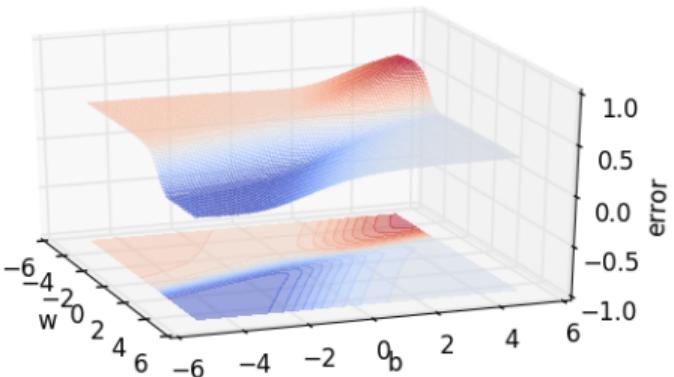


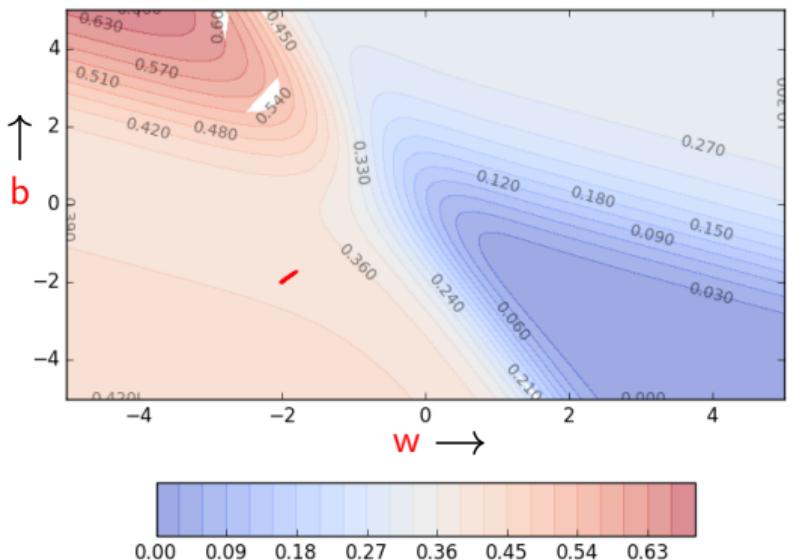
Gradient descent on the error surface



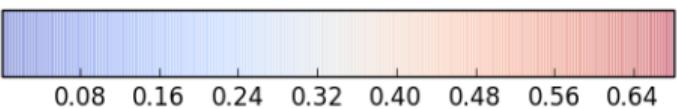
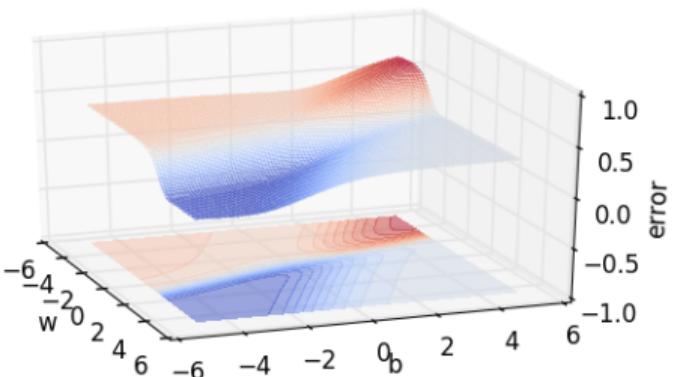


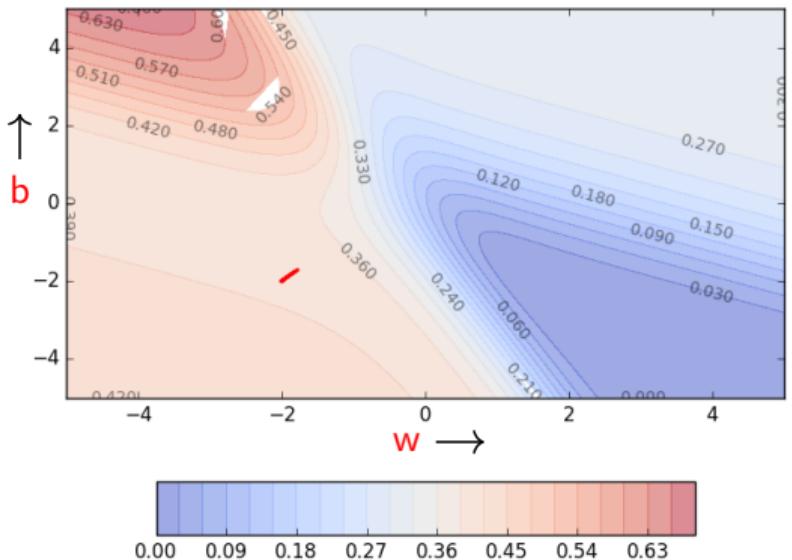
Gradient descent on the error surface



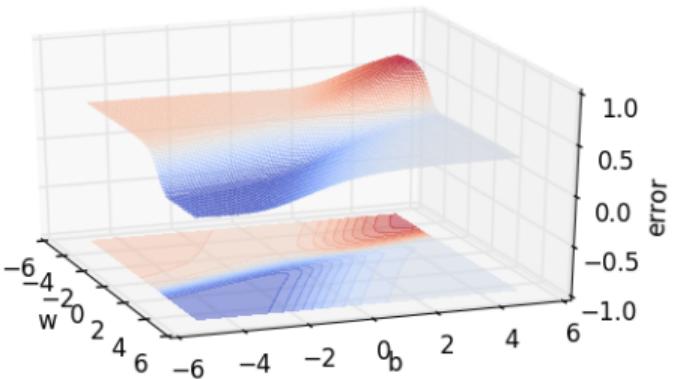


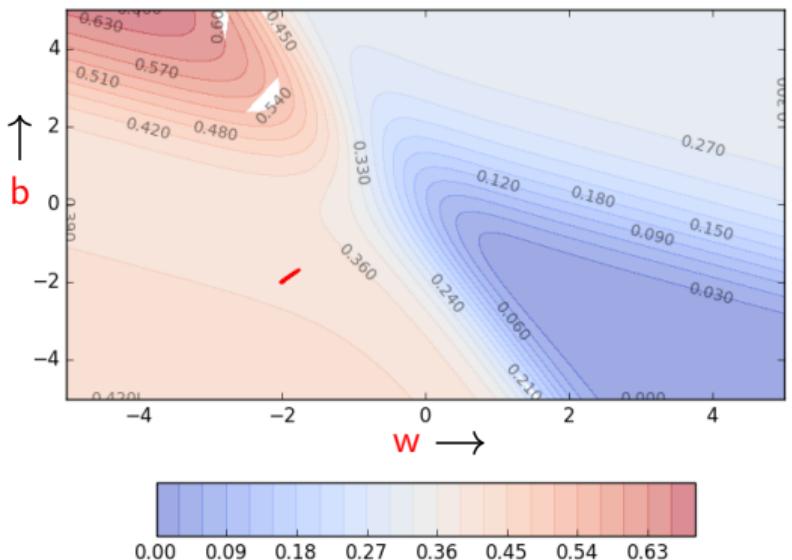
Gradient descent on the error surface



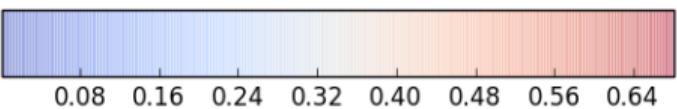
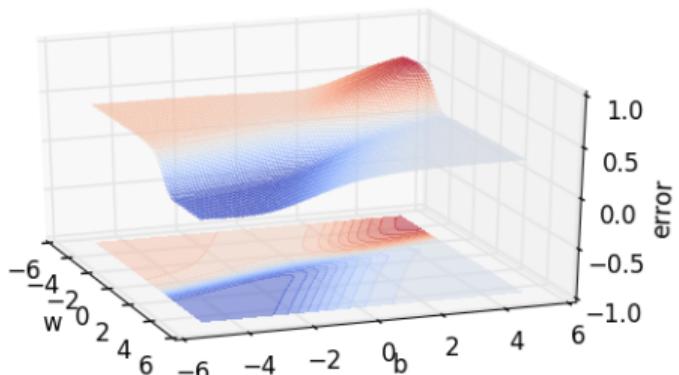


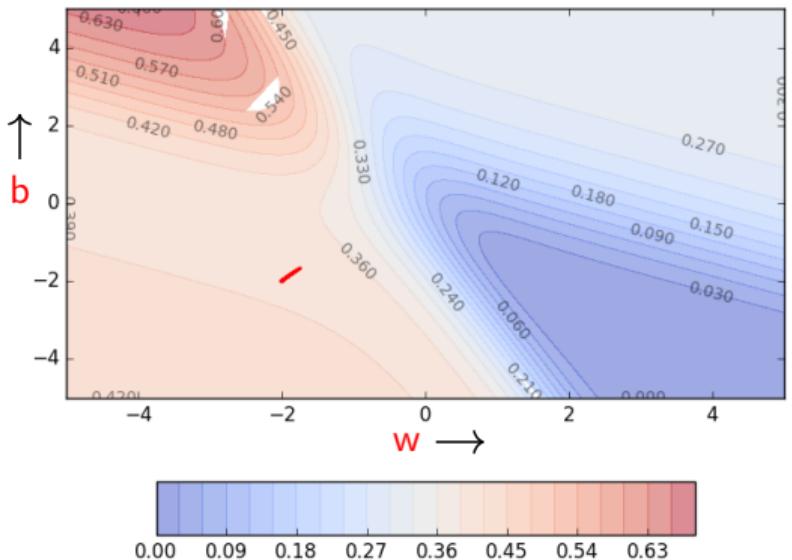
Gradient descent on the error surface



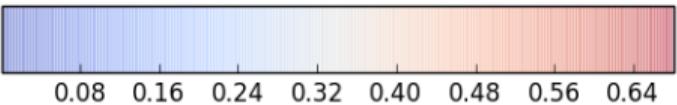
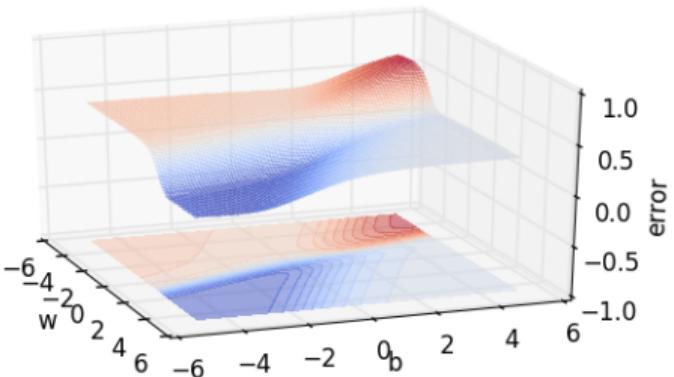


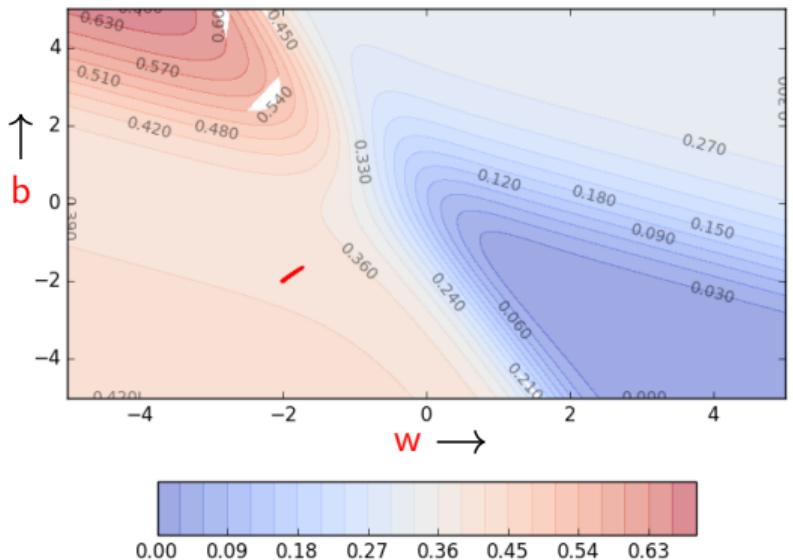
Gradient descent on the error surface



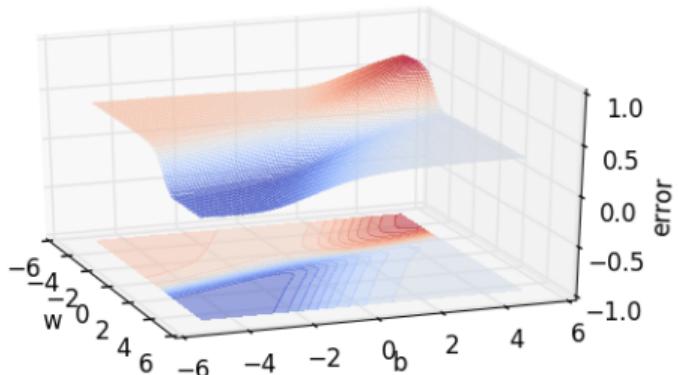


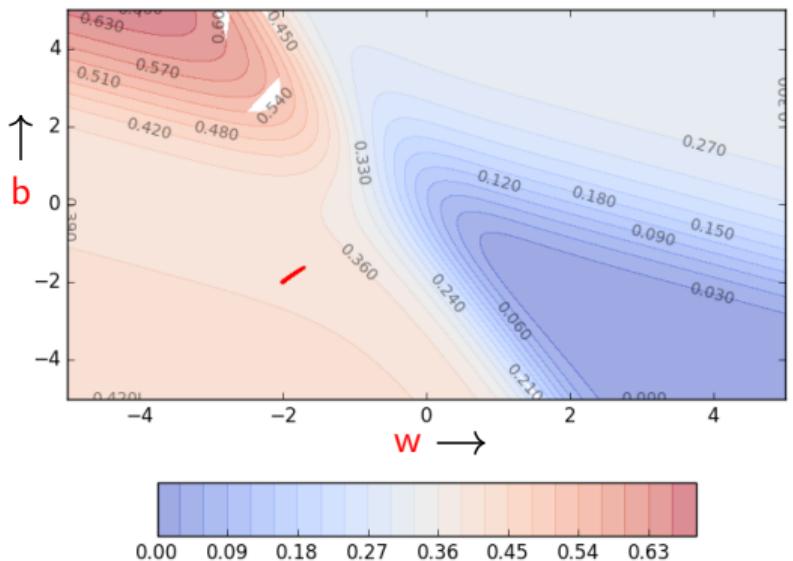
Gradient descent on the error surface



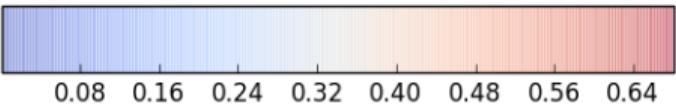
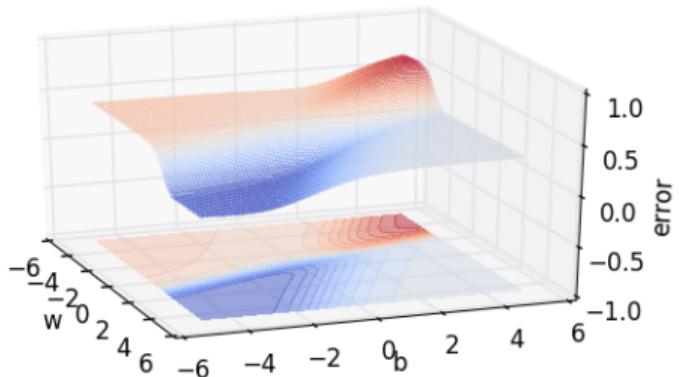


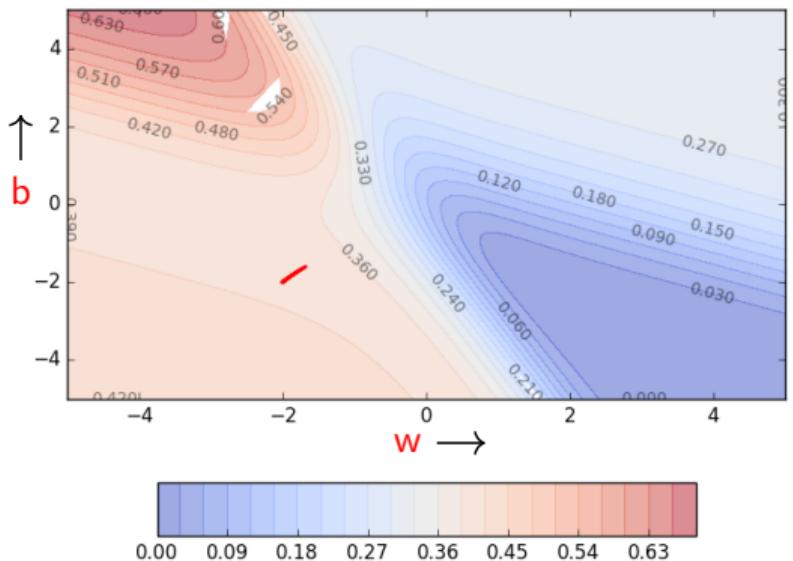
Gradient descent on the error surface



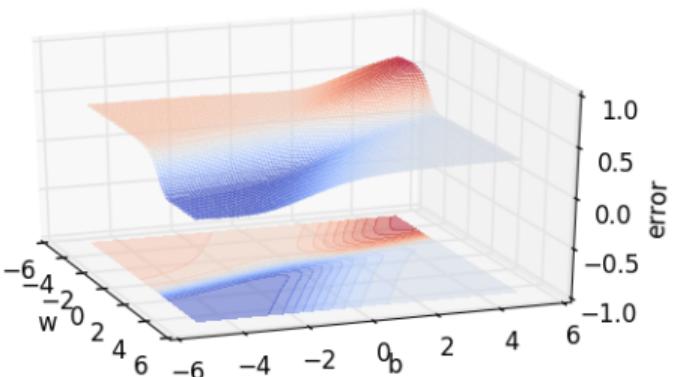


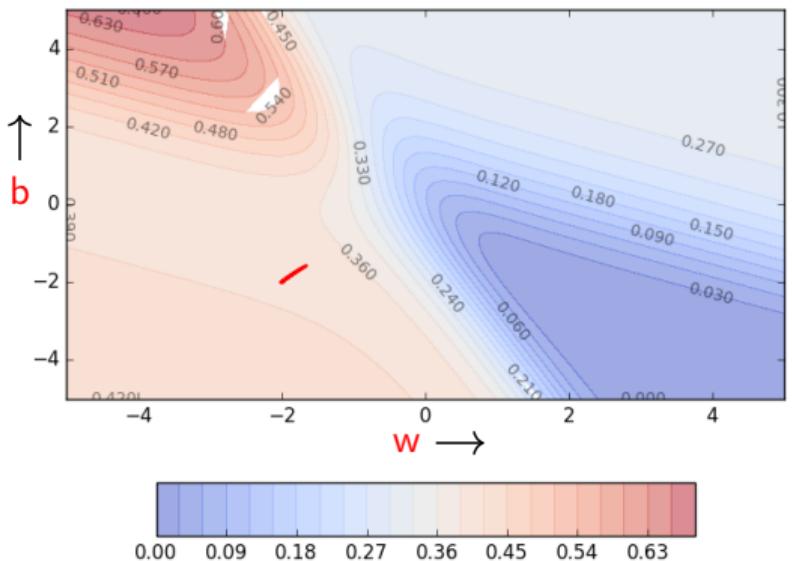
Gradient descent on the error surface



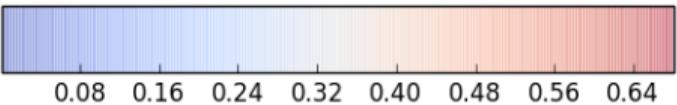
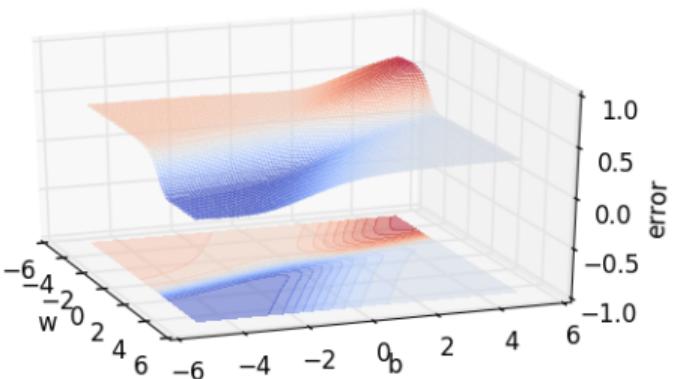


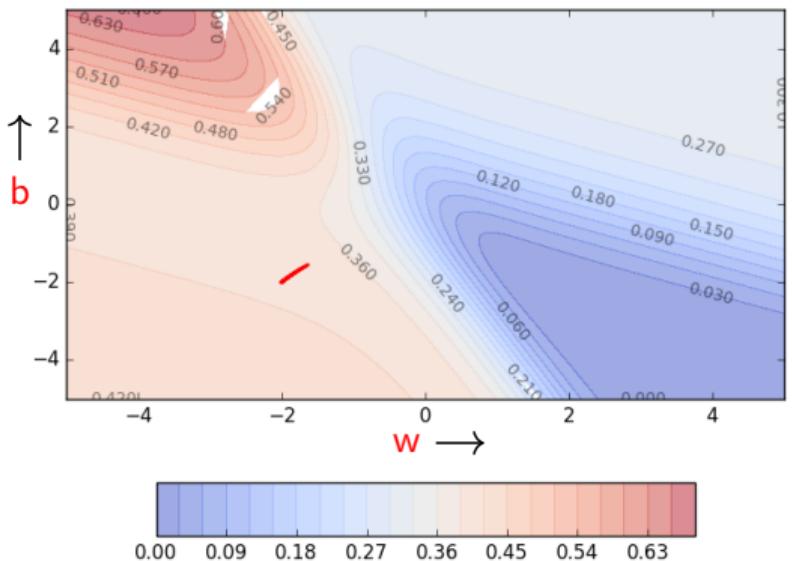
Gradient descent on the error surface



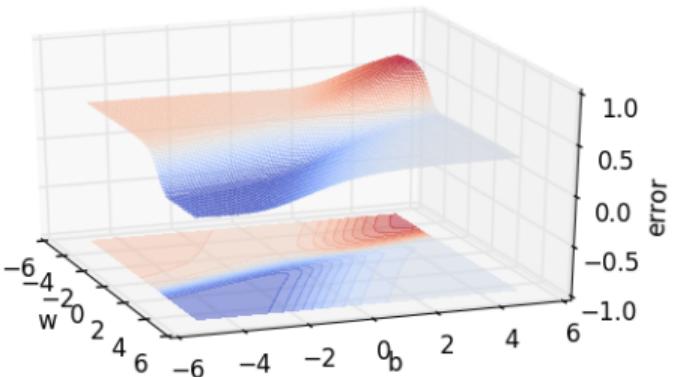


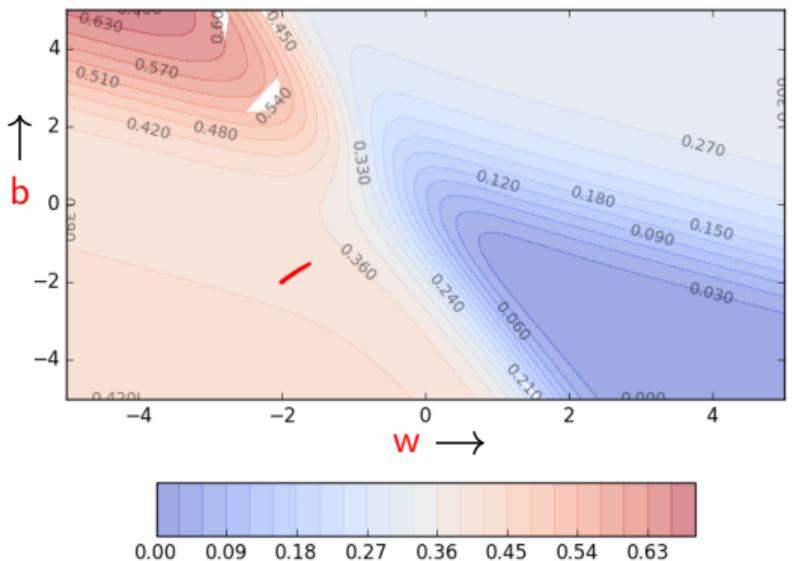
Gradient descent on the error surface



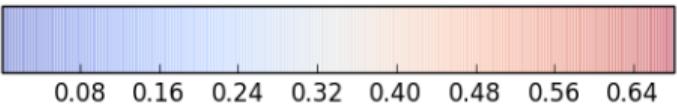
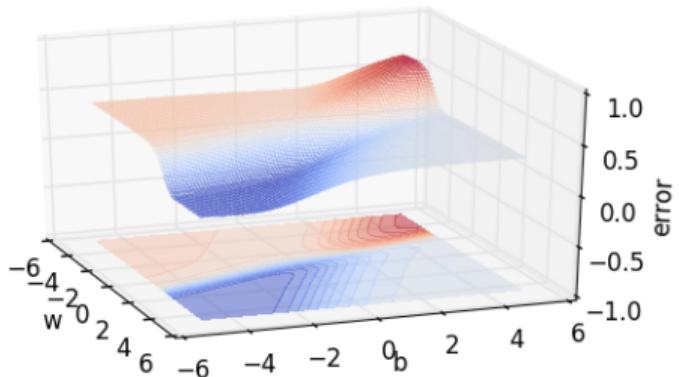


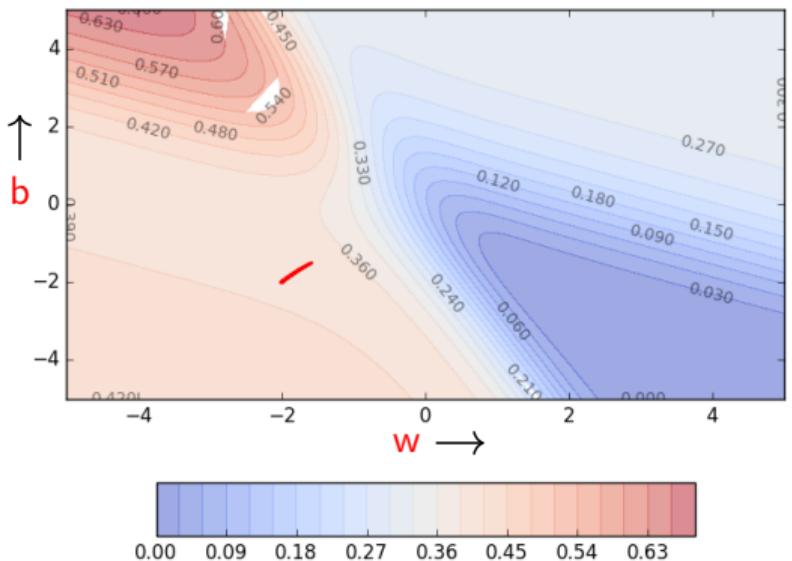
Gradient descent on the error surface



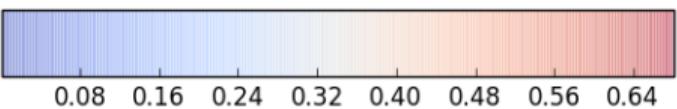
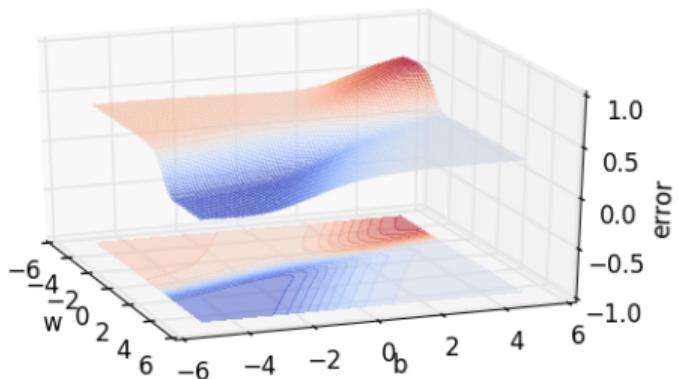


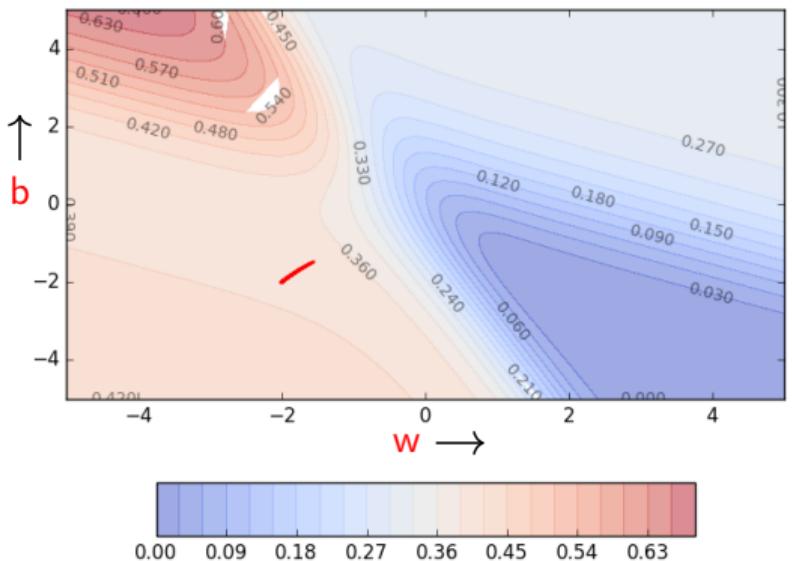
Gradient descent on the error surface



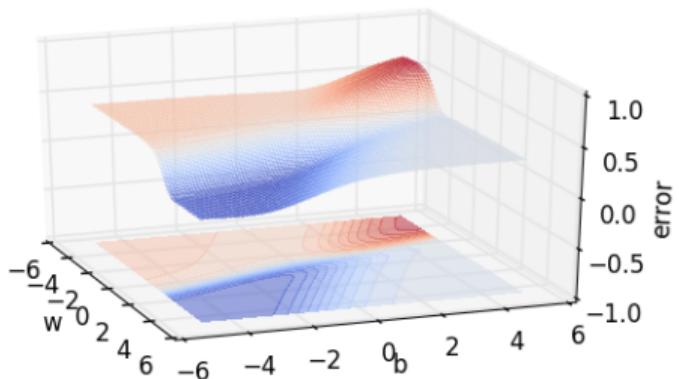


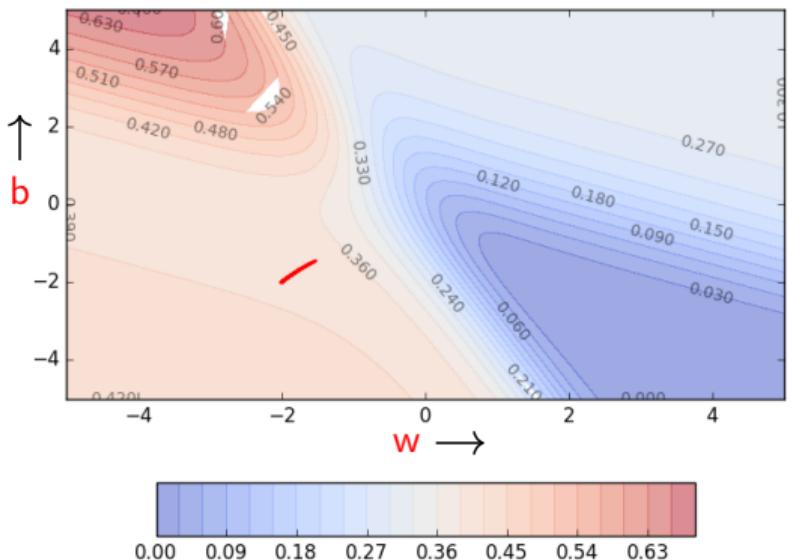
Gradient descent on the error surface



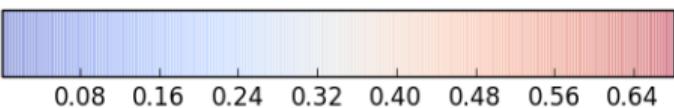
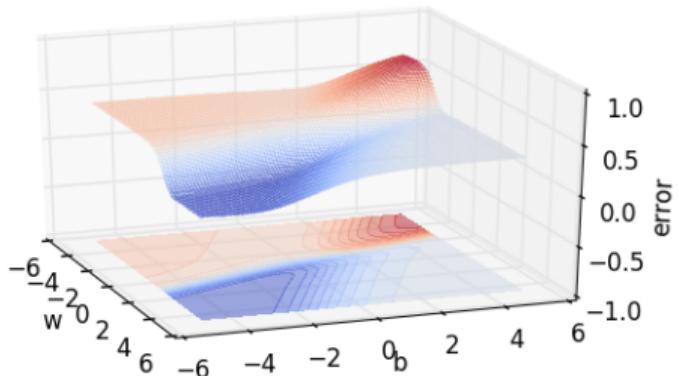


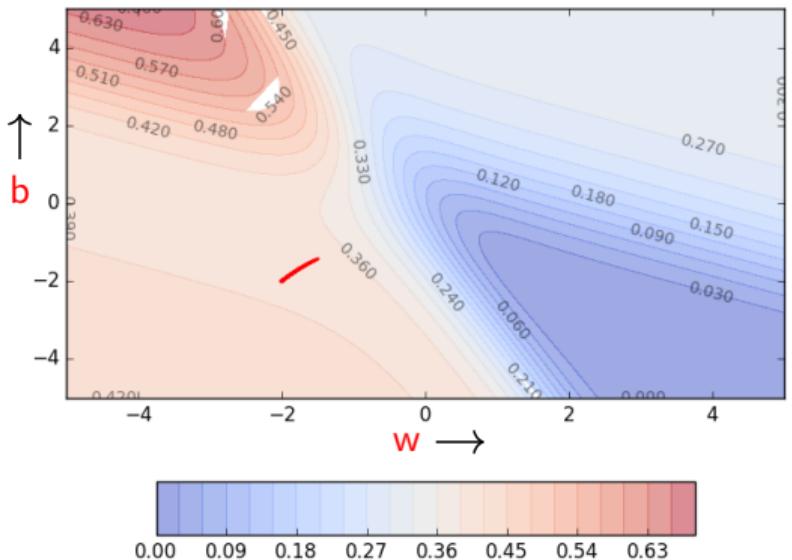
Gradient descent on the error surface



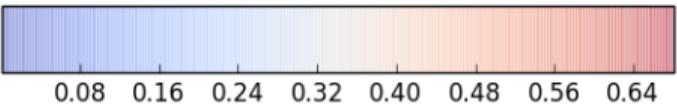
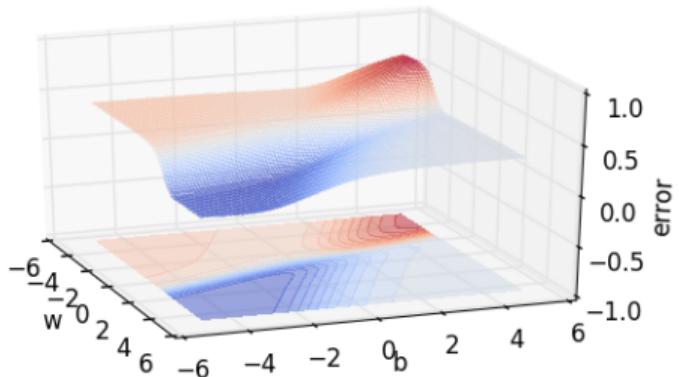


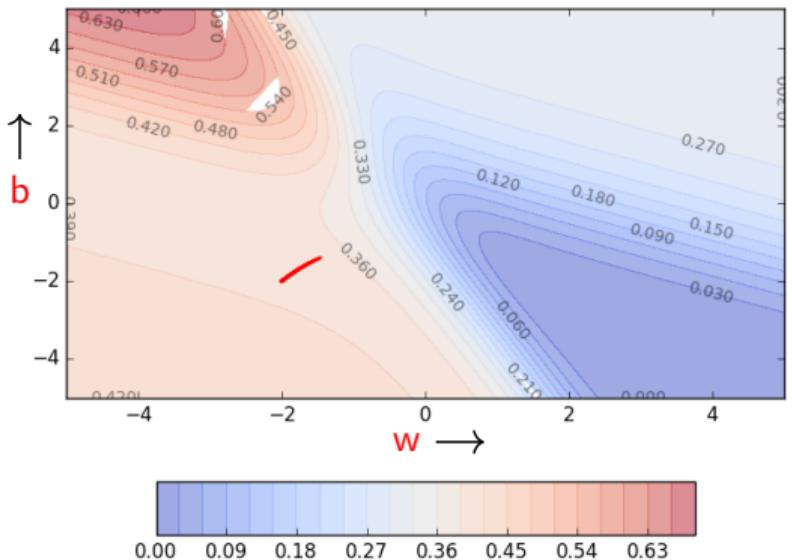
Gradient descent on the error surface



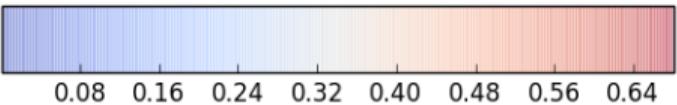
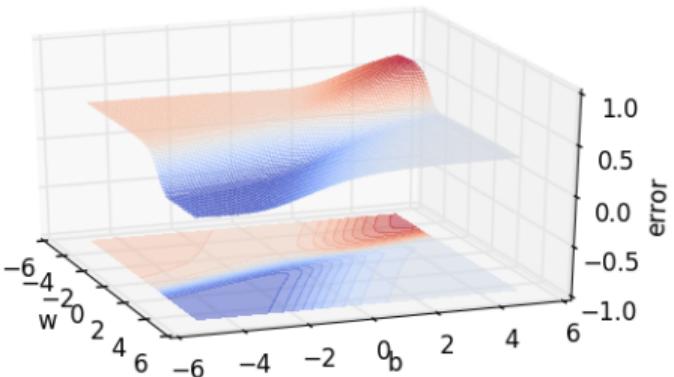


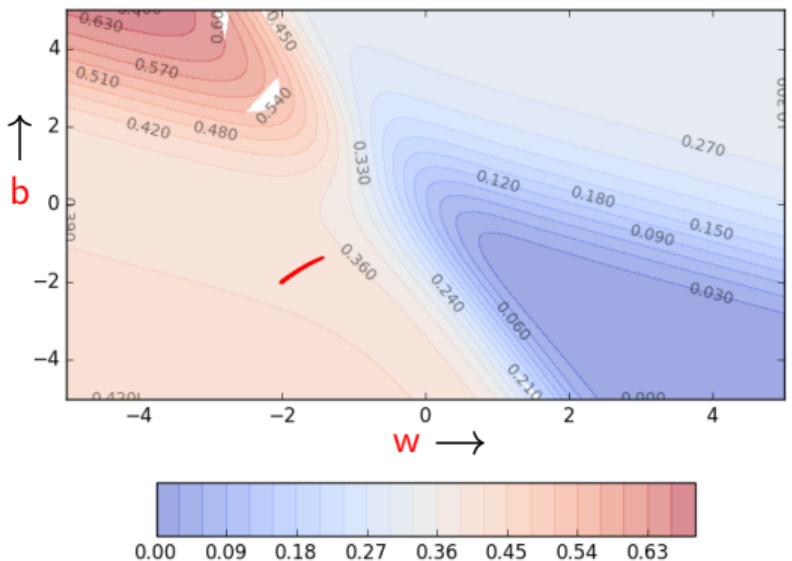
Gradient descent on the error surface



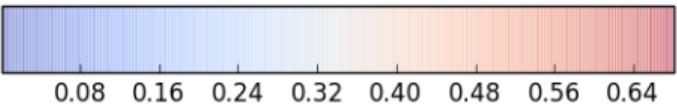
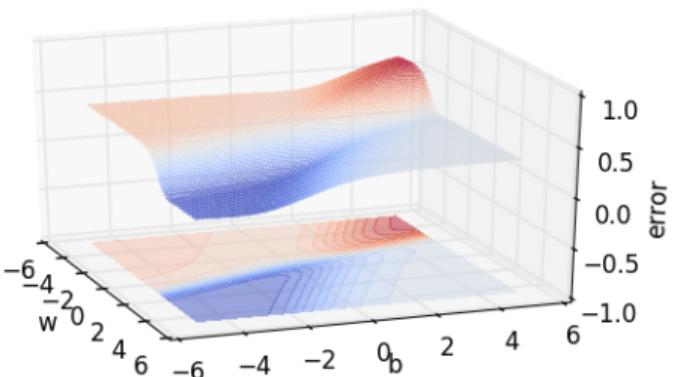


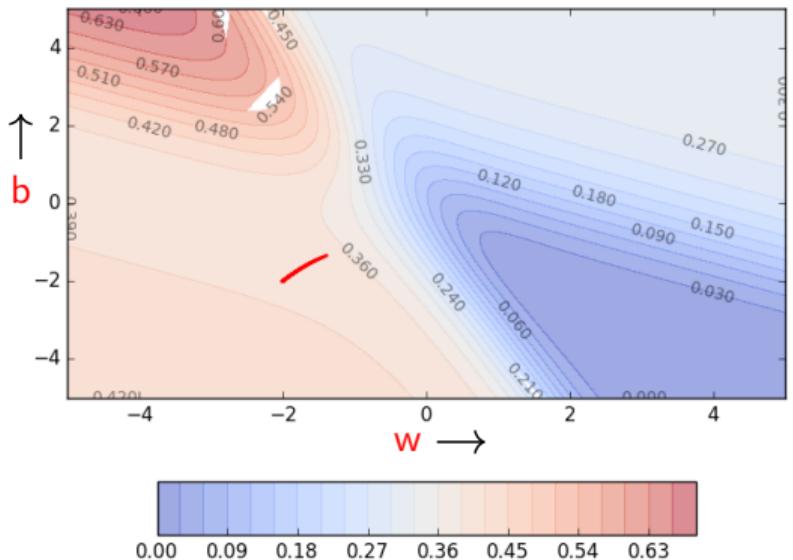
Gradient descent on the error surface



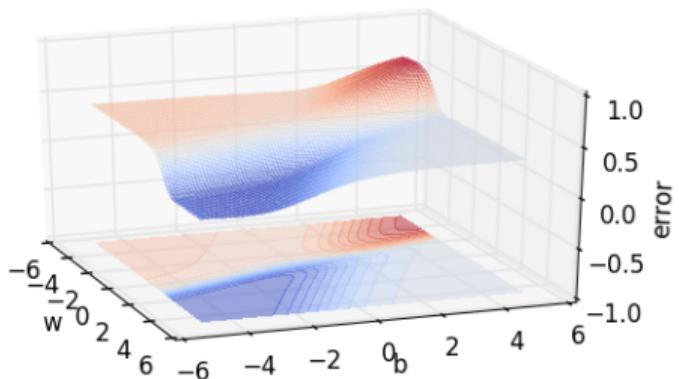


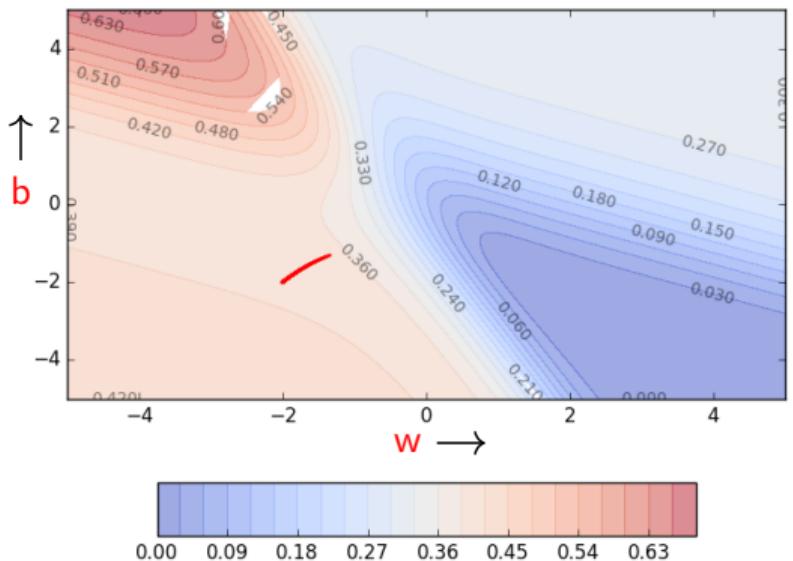
Gradient descent on the error surface



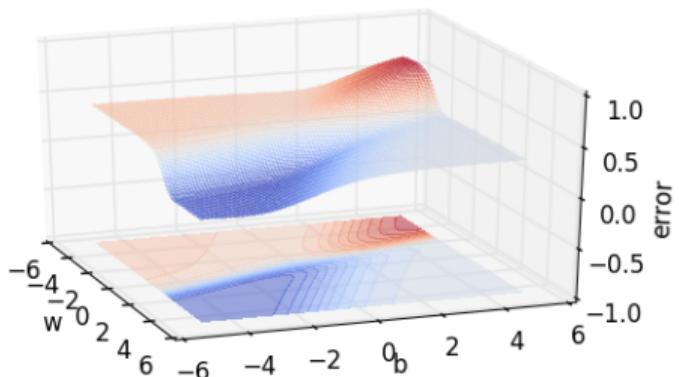


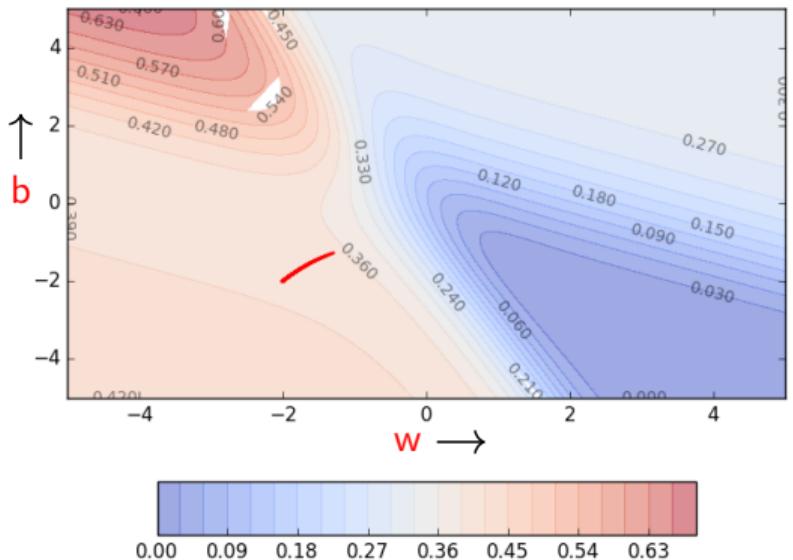
Gradient descent on the error surface



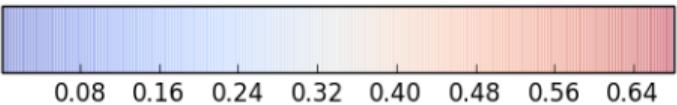
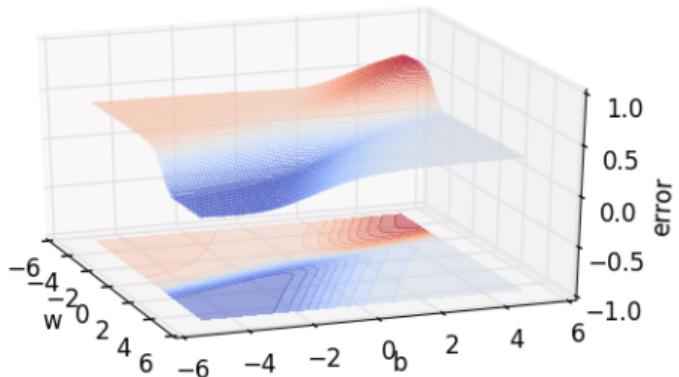


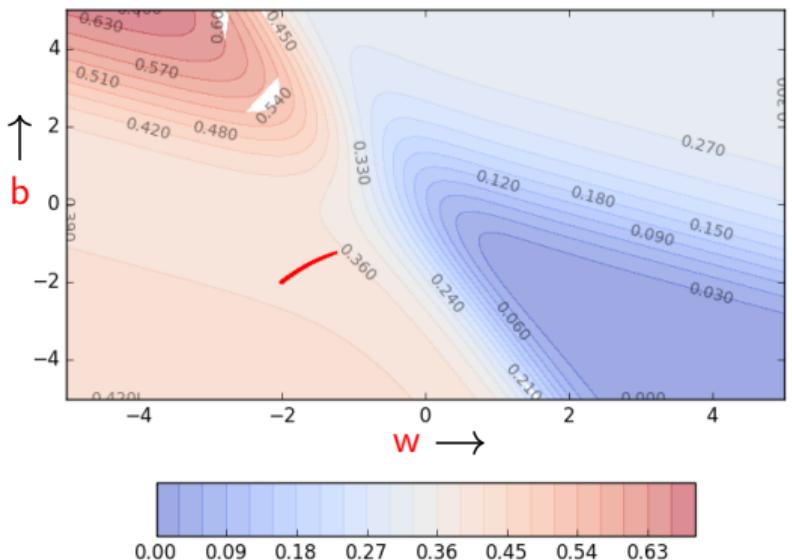
Gradient descent on the error surface



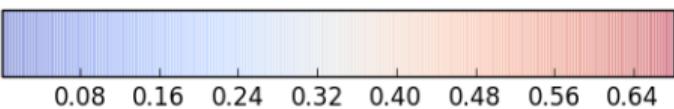
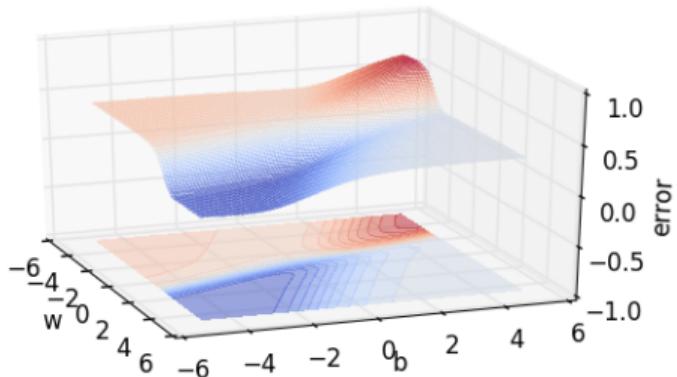


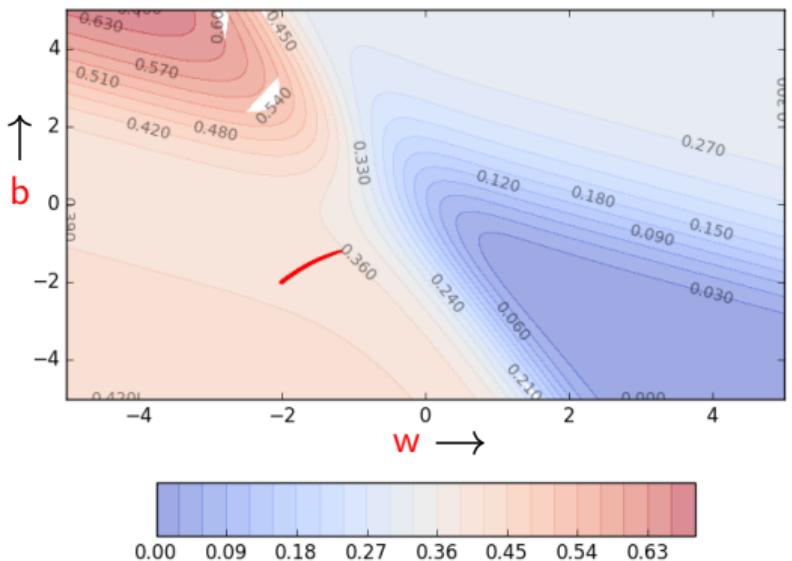
Gradient descent on the error surface



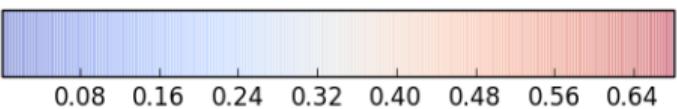
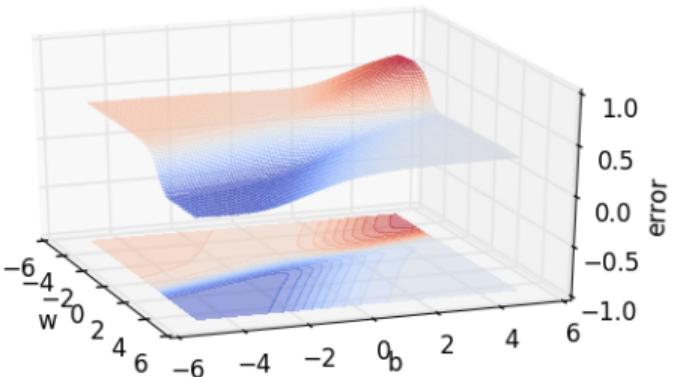


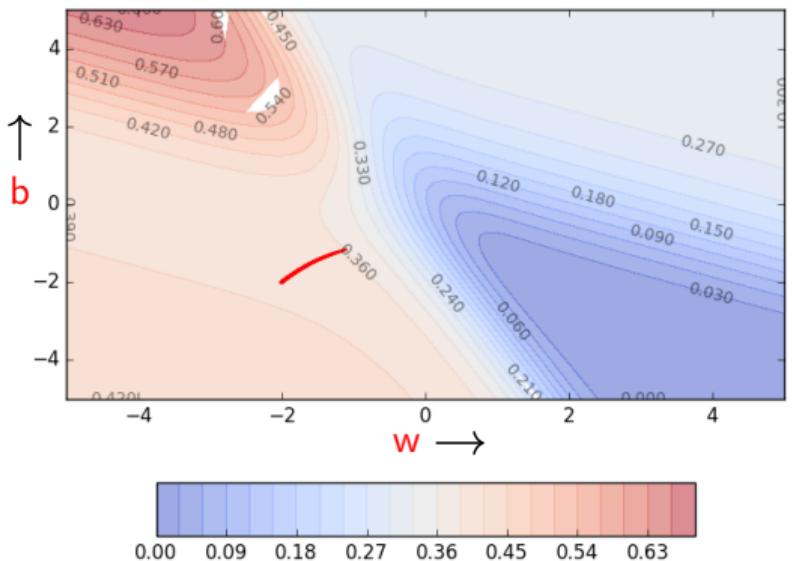
Gradient descent on the error surface



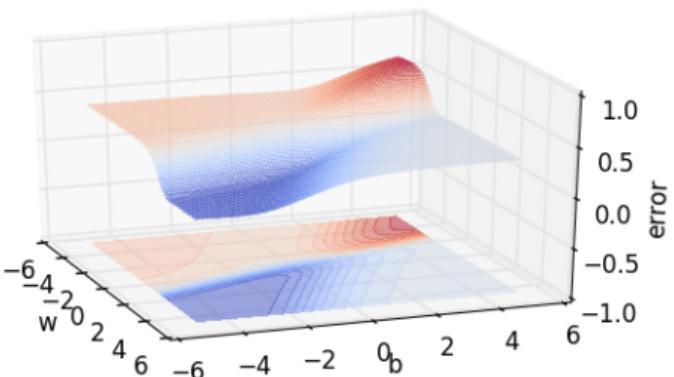


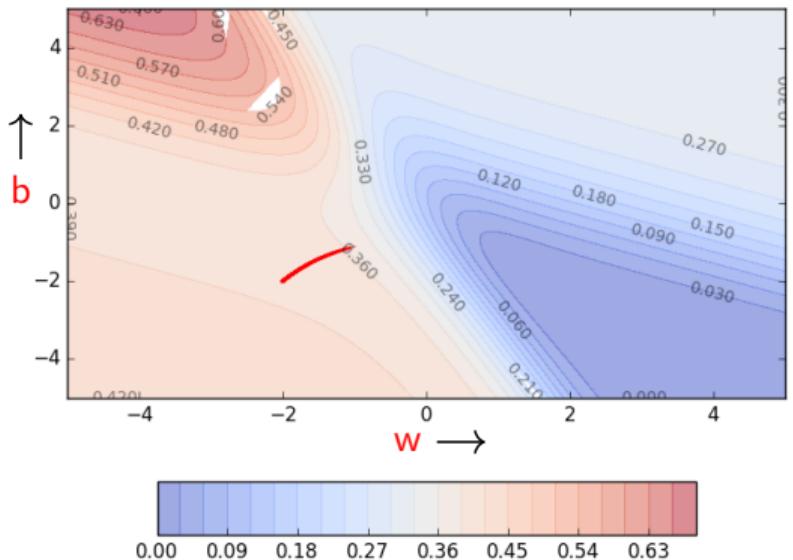
Gradient descent on the error surface



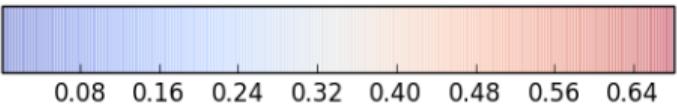
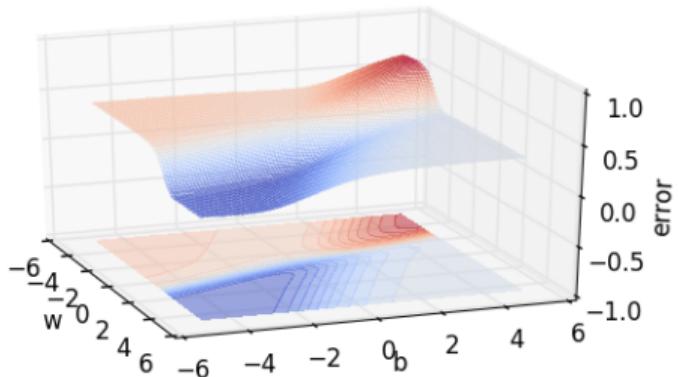


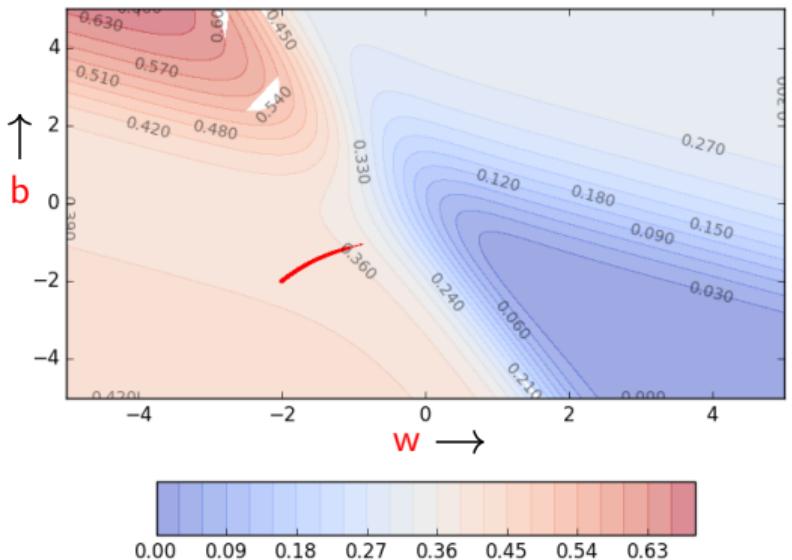
Gradient descent on the error surface



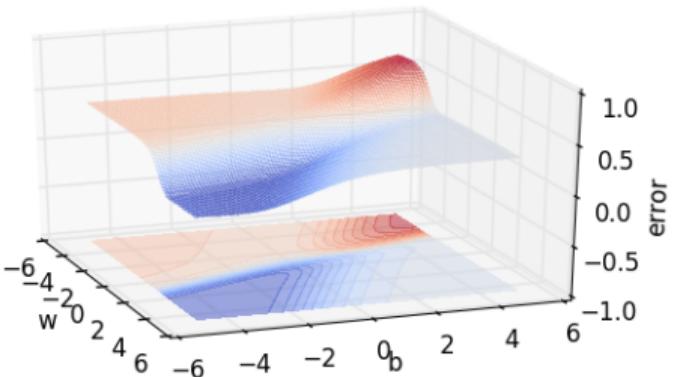


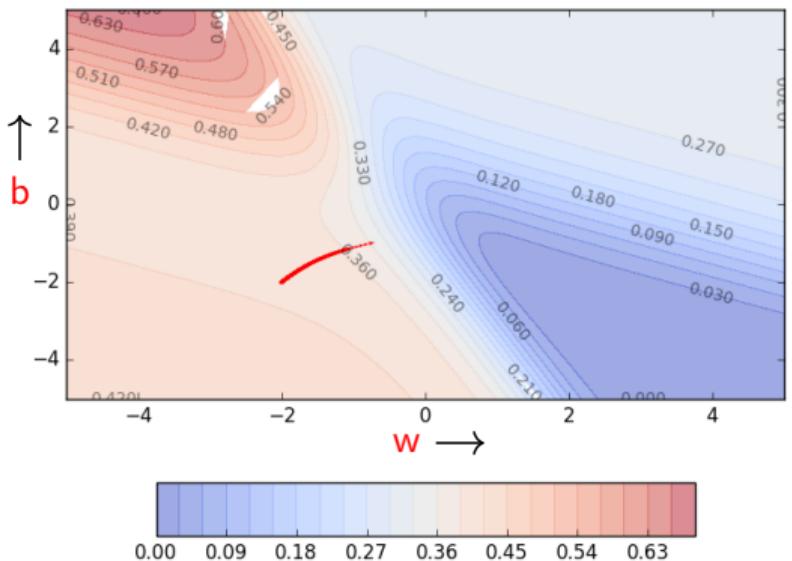
Gradient descent on the error surface



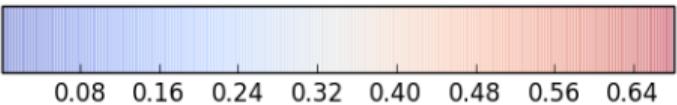
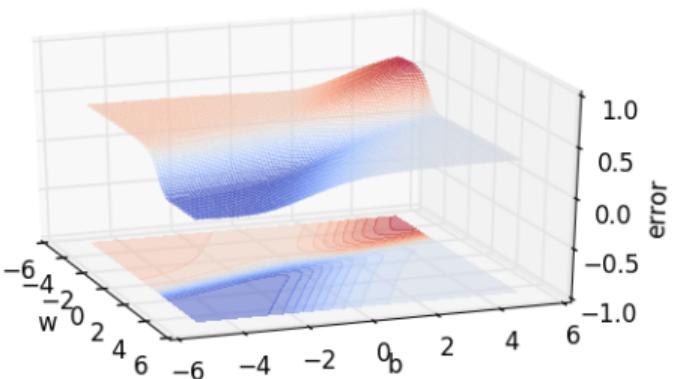


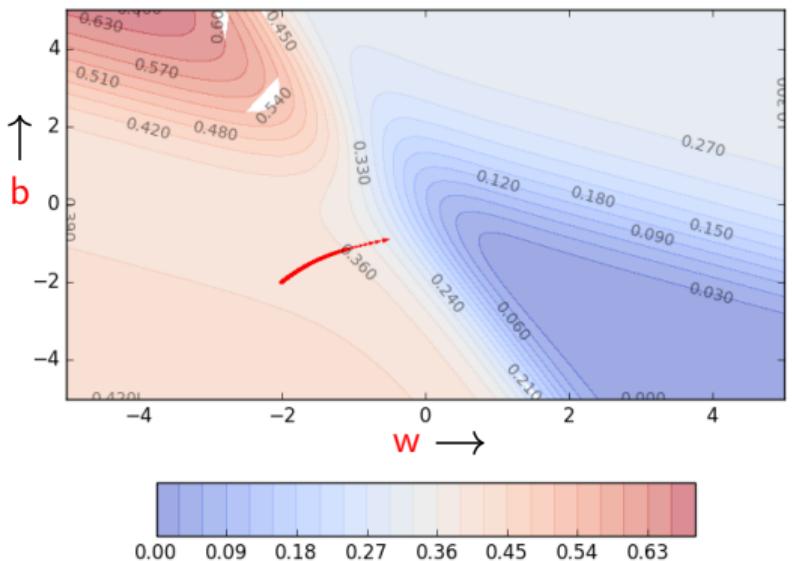
Gradient descent on the error surface



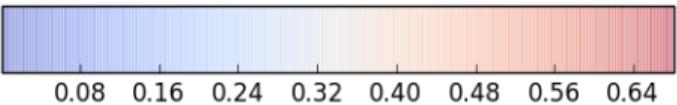
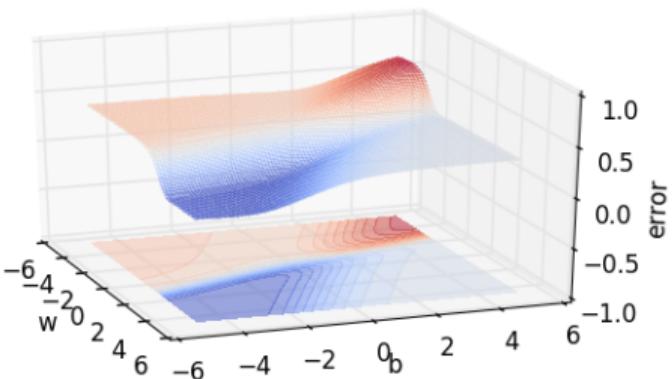


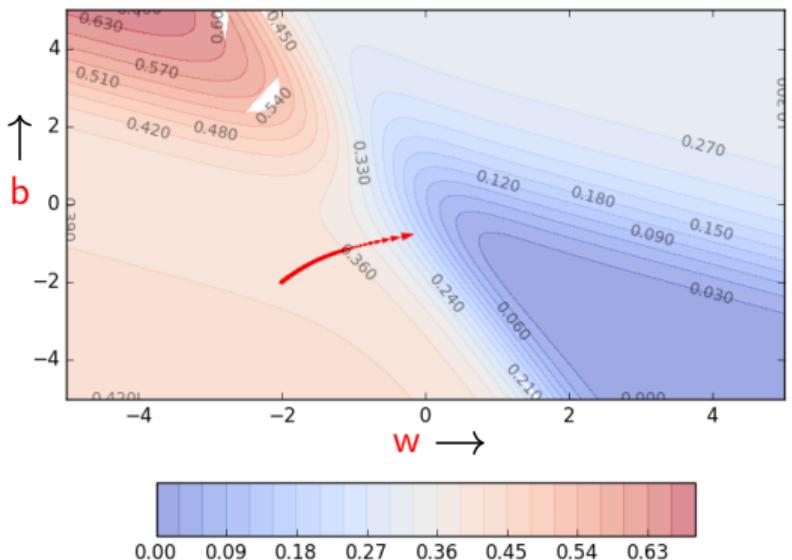
Gradient descent on the error surface



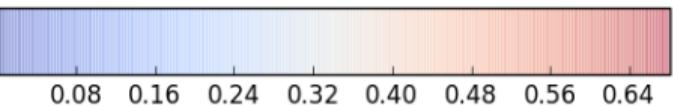
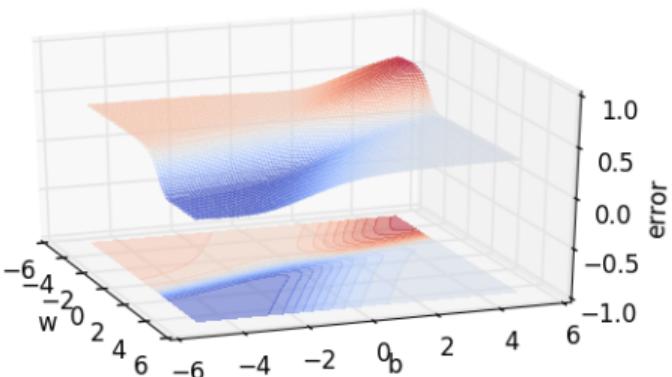


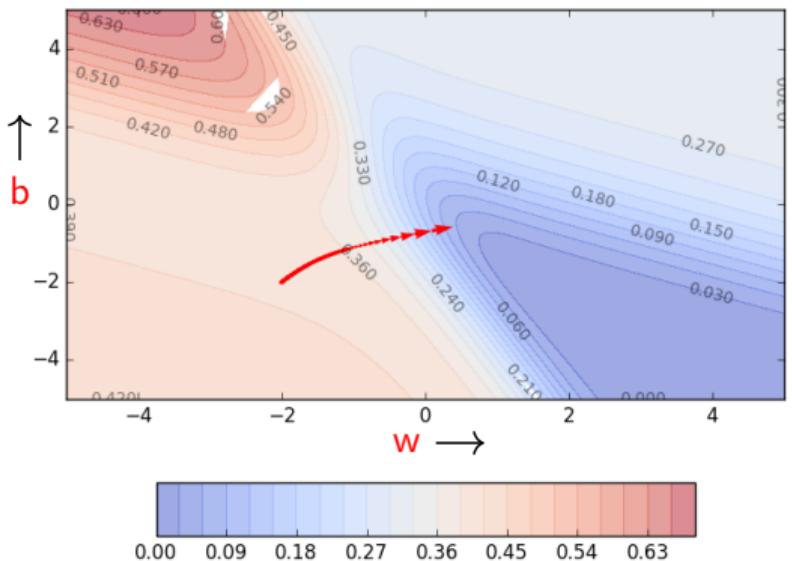
Gradient descent on the error surface



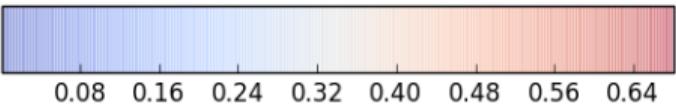
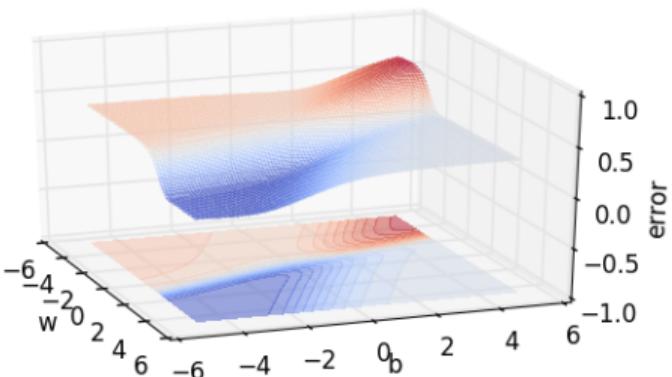


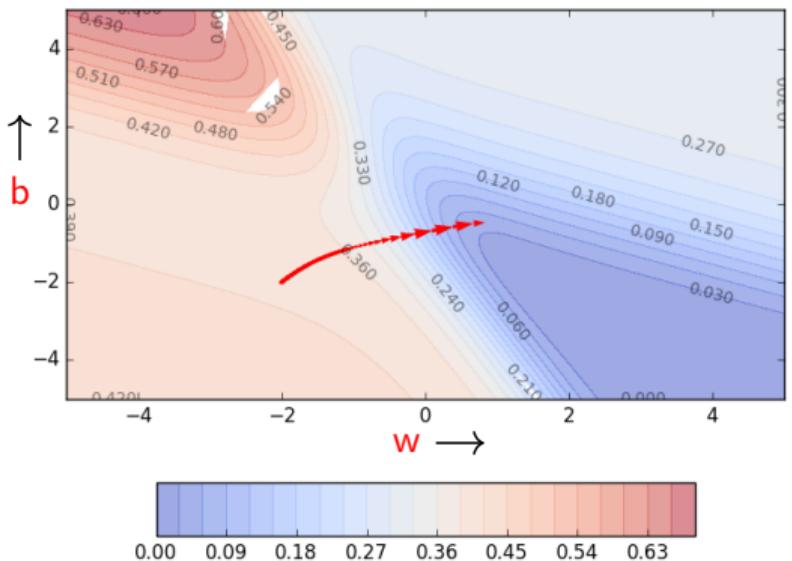
Gradient descent on the error surface



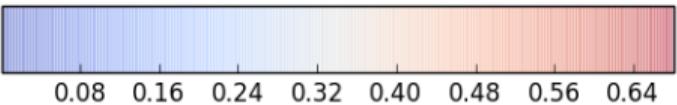
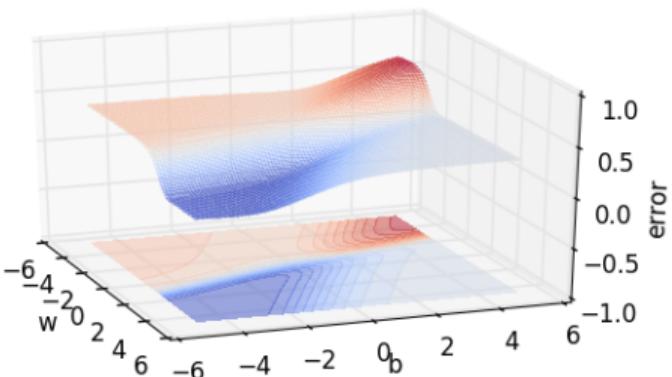


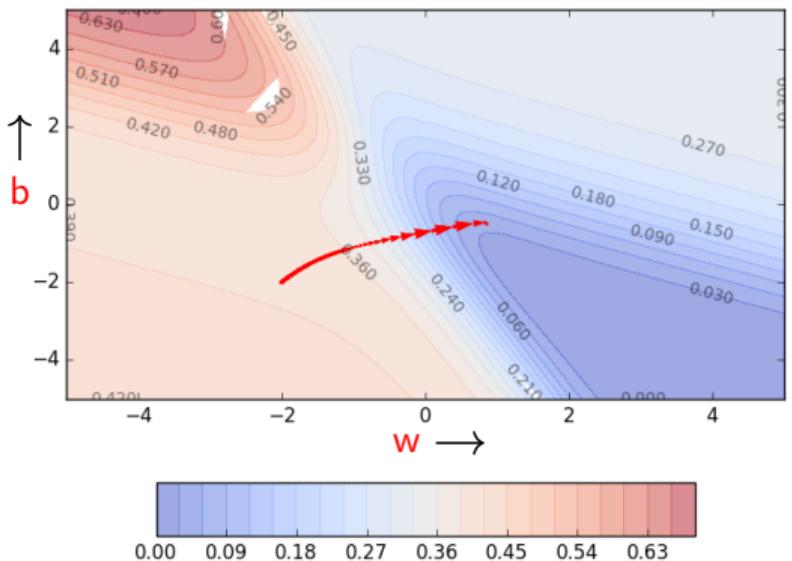
Gradient descent on the error surface



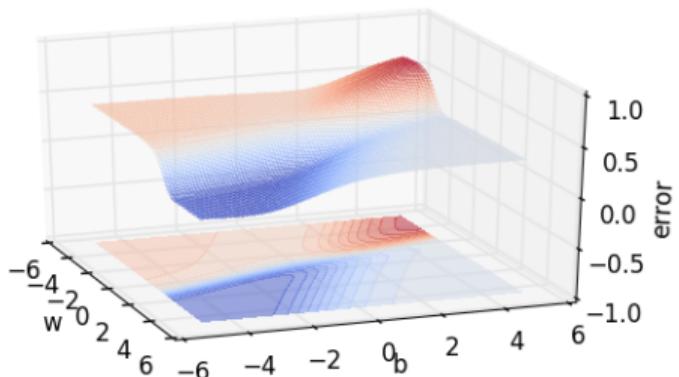


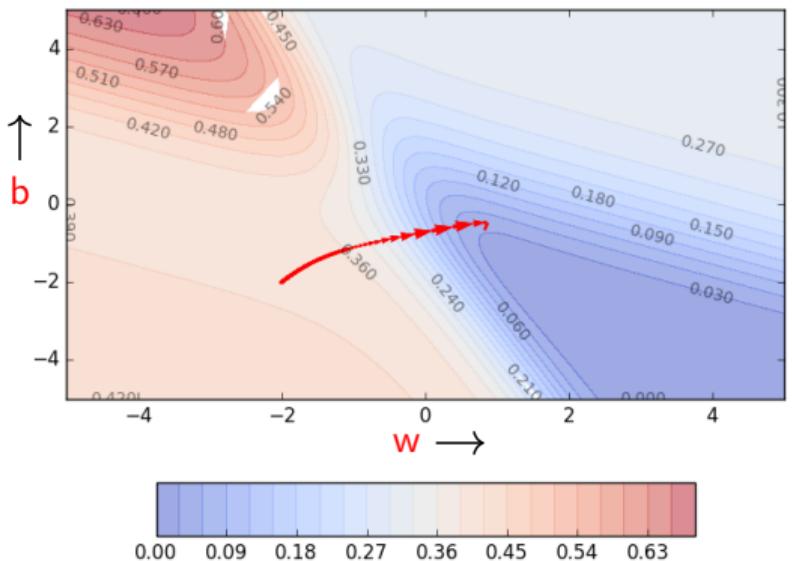
Gradient descent on the error surface



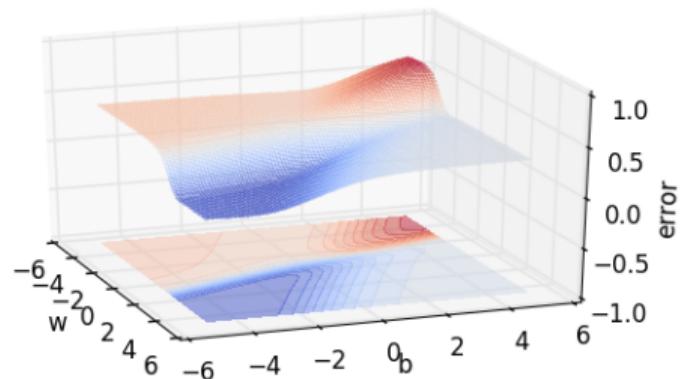


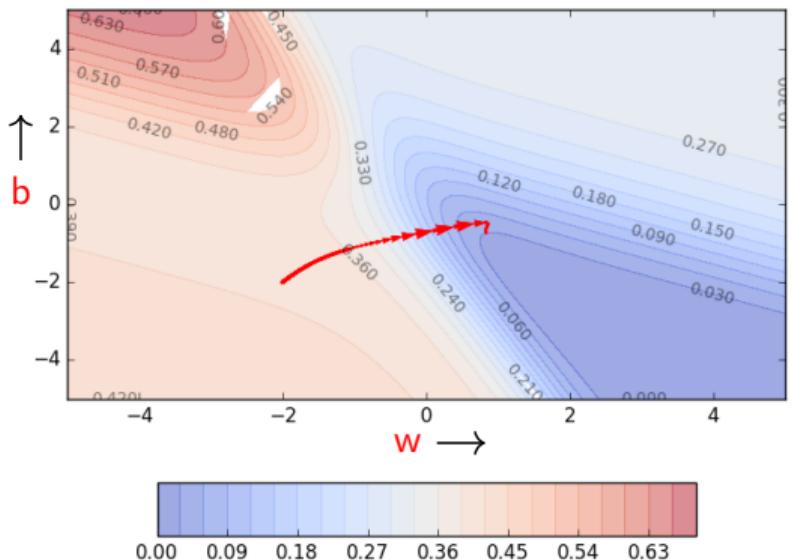
Gradient descent on the error surface



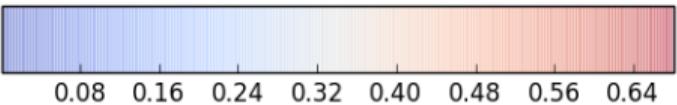
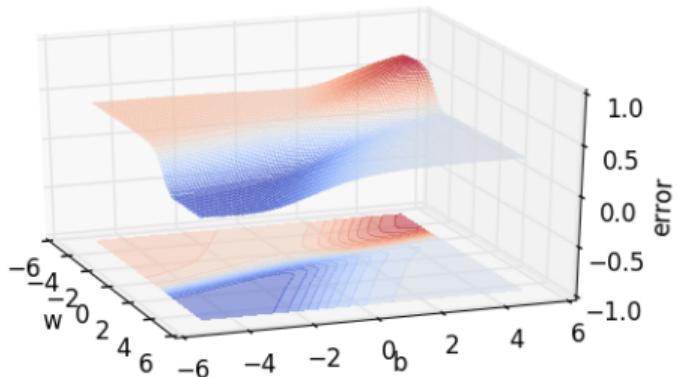


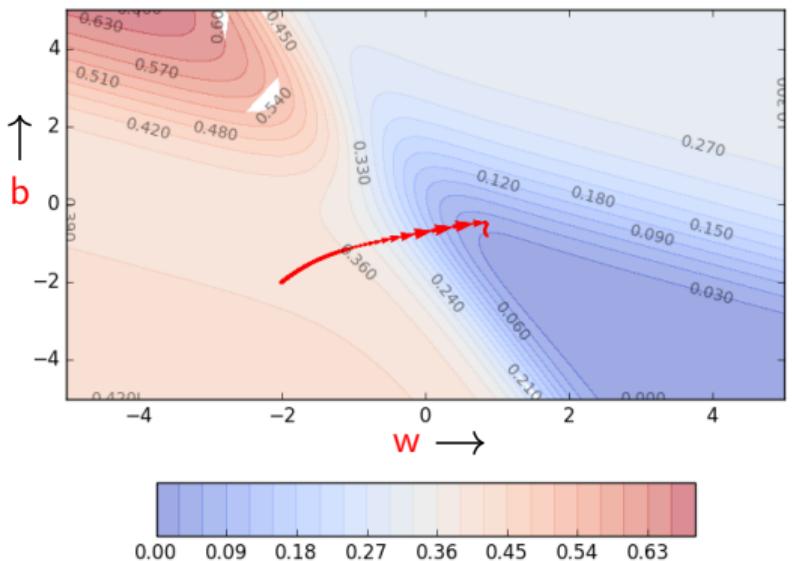
Gradient descent on the error surface



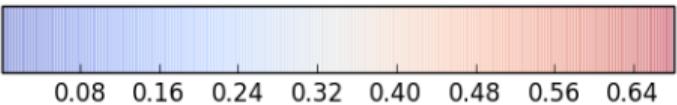
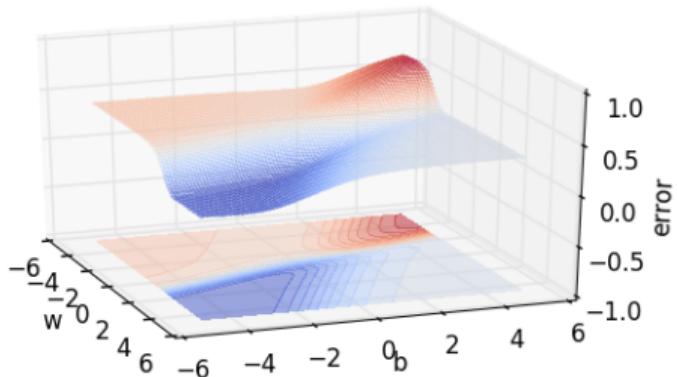


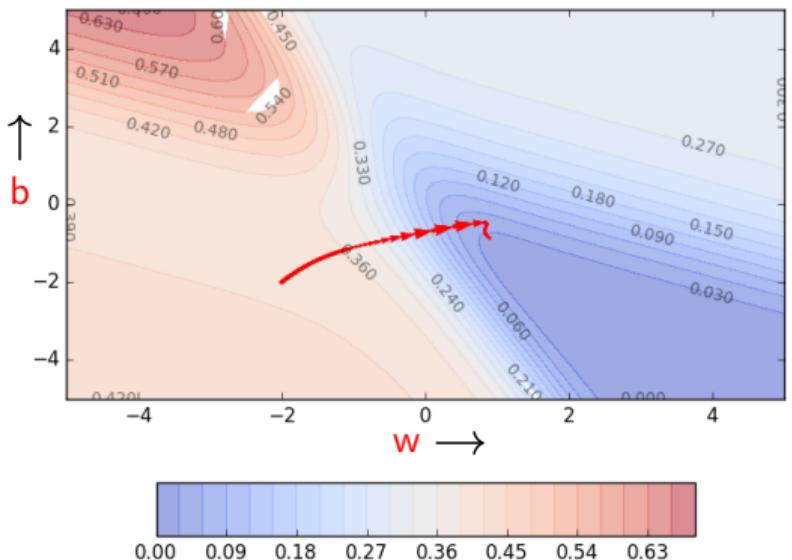
Gradient descent on the error surface



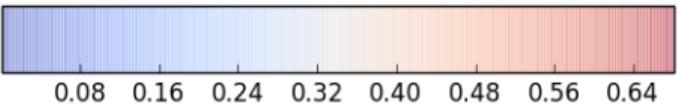
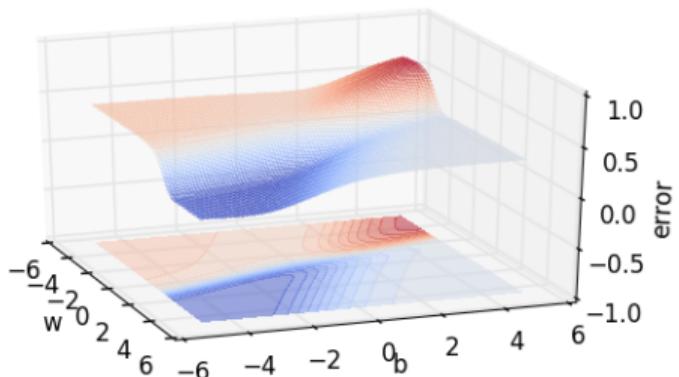


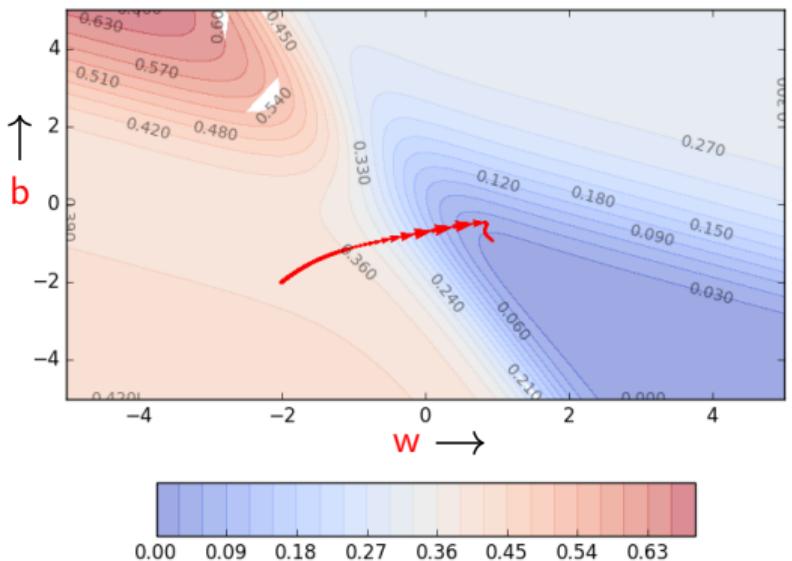
Gradient descent on the error surface



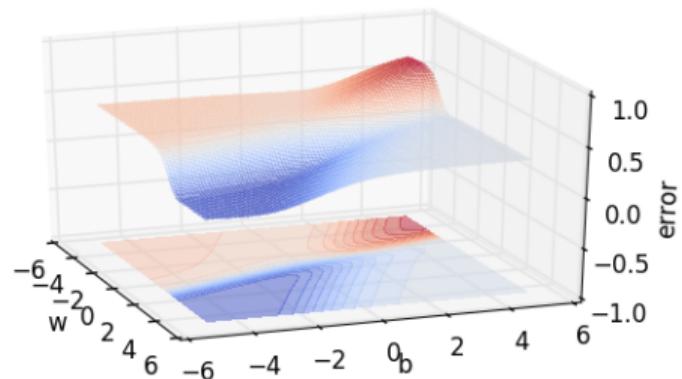


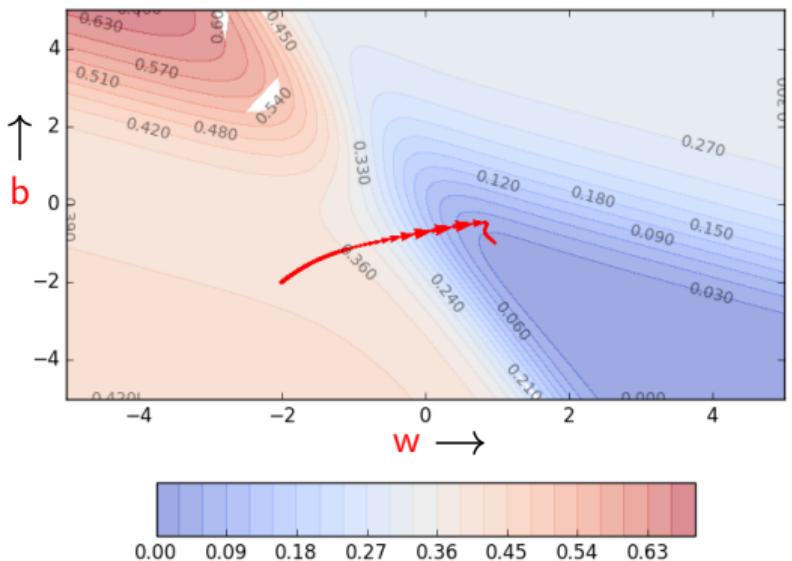
Gradient descent on the error surface



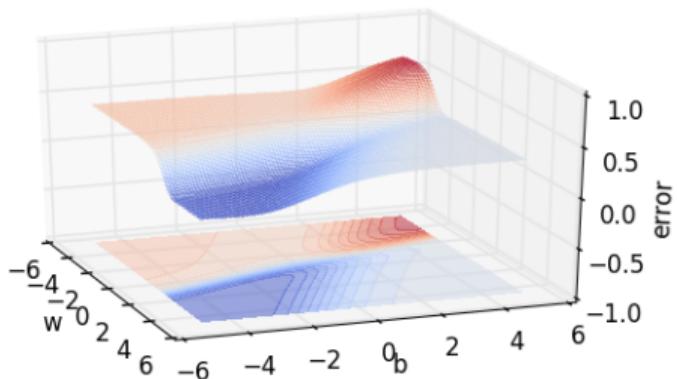


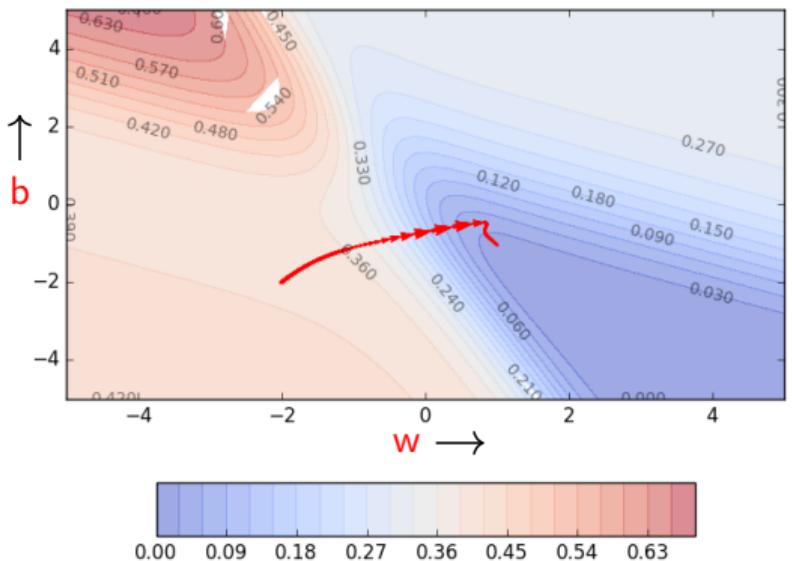
Gradient descent on the error surface



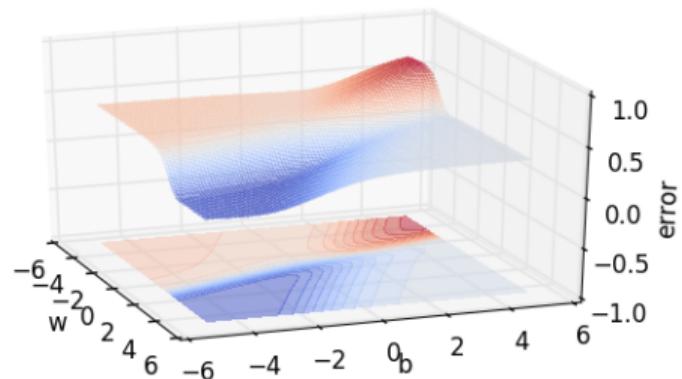


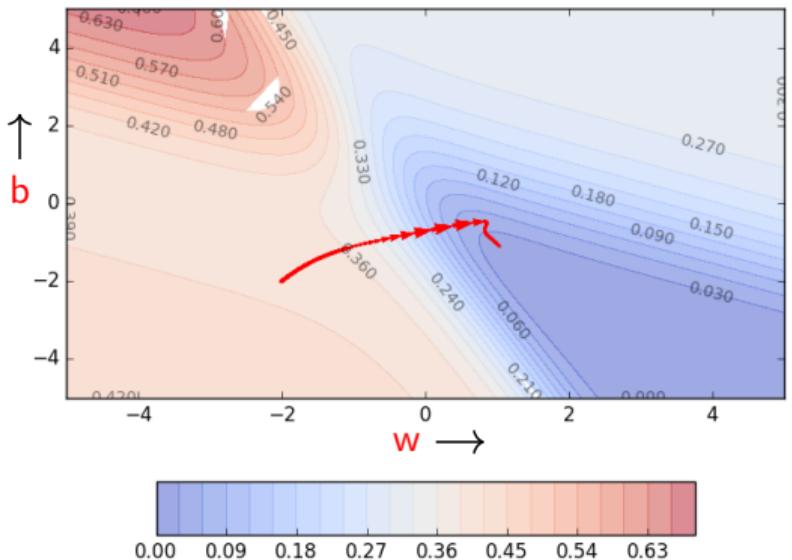
Gradient descent on the error surface



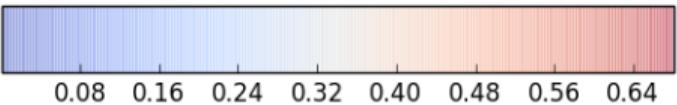
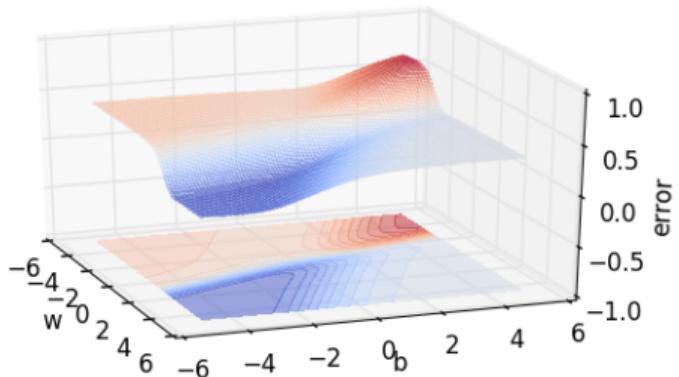


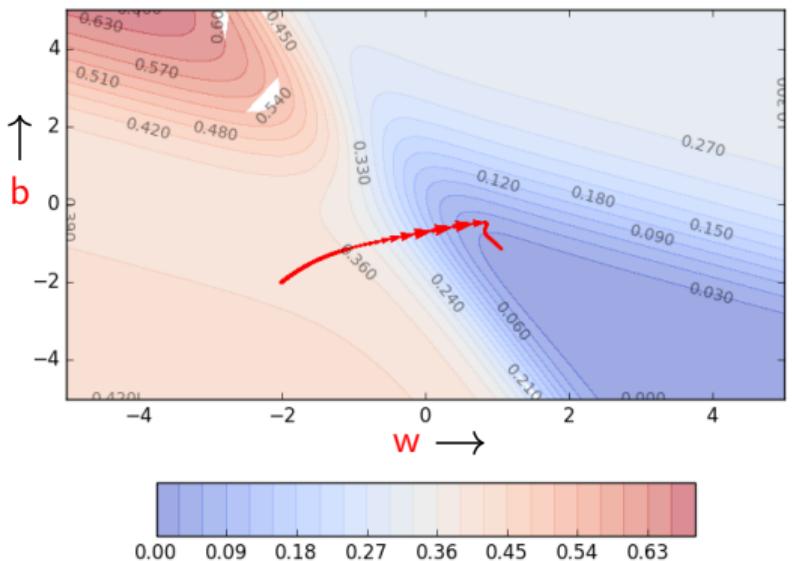
Gradient descent on the error surface



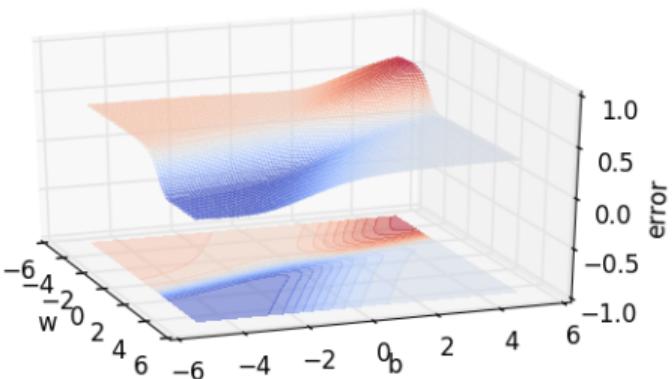


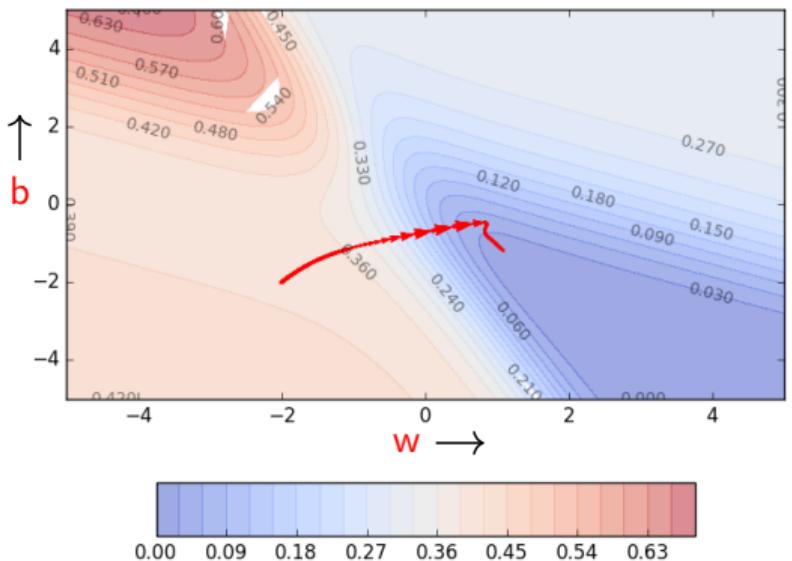
Gradient descent on the error surface



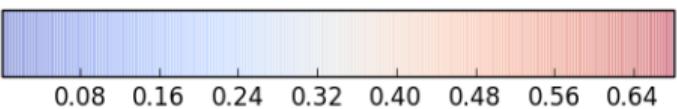
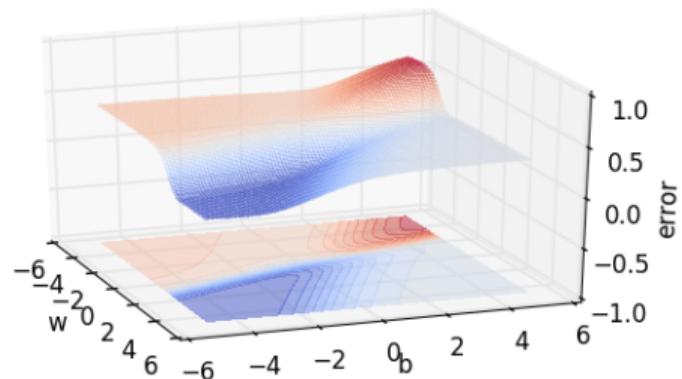


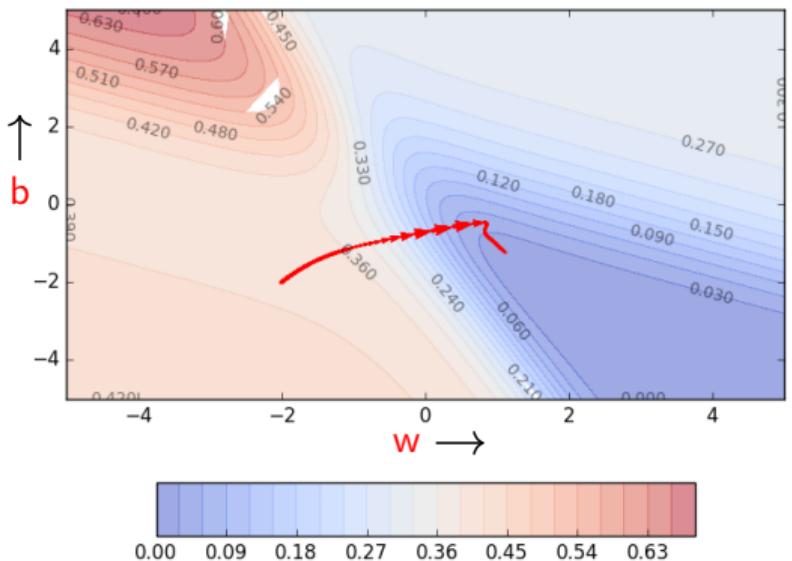
Gradient descent on the error surface



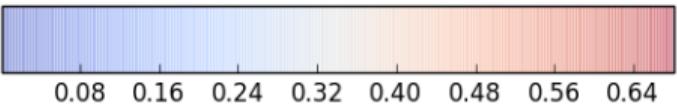
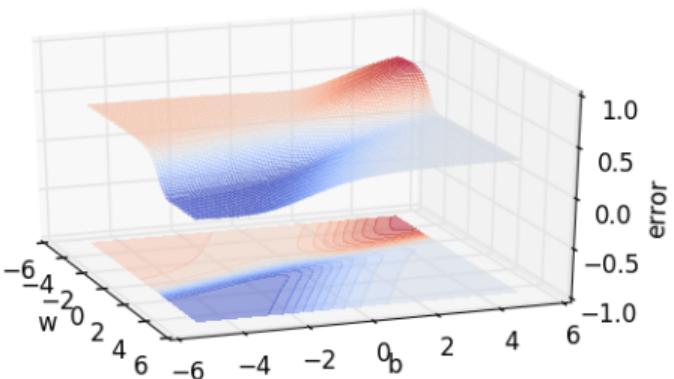


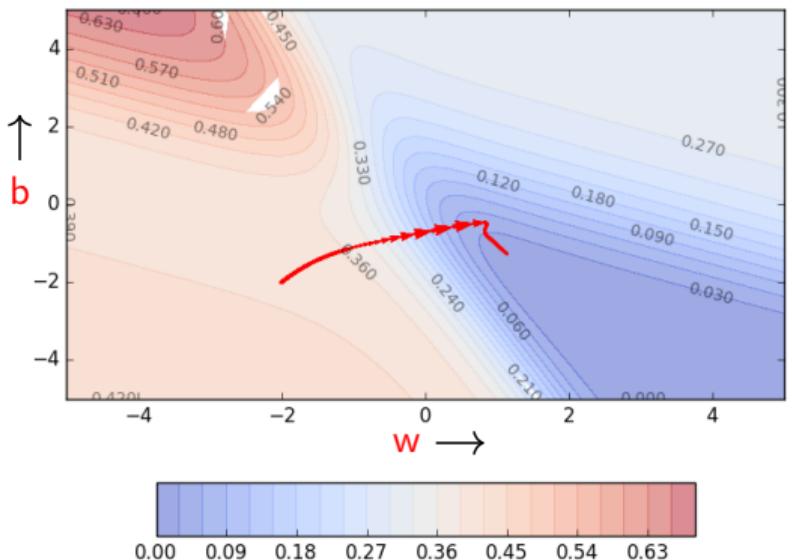
Gradient descent on the error surface



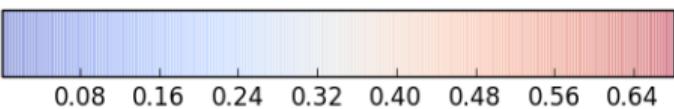
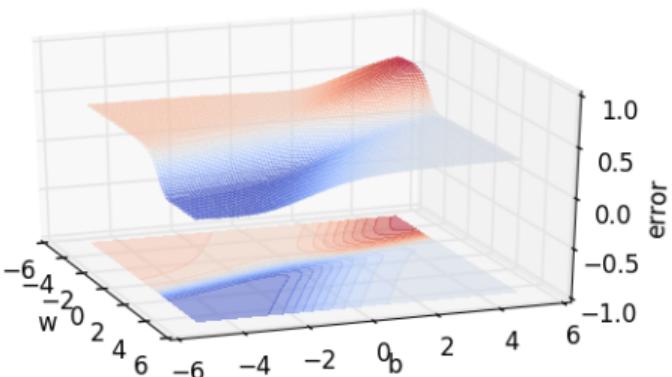


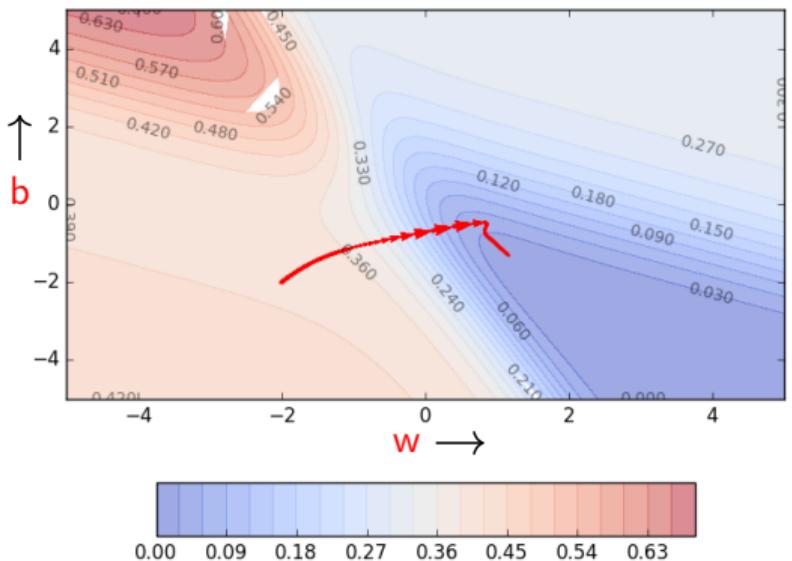
Gradient descent on the error surface



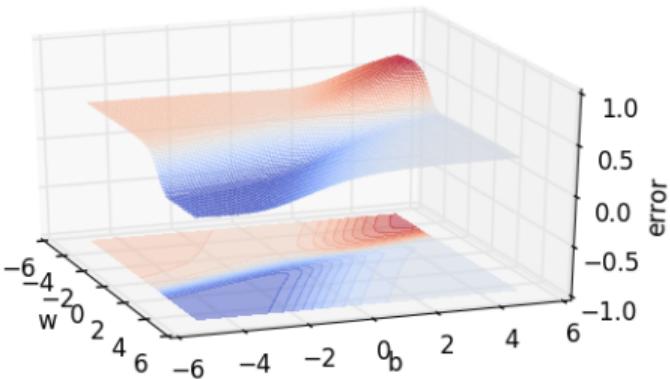


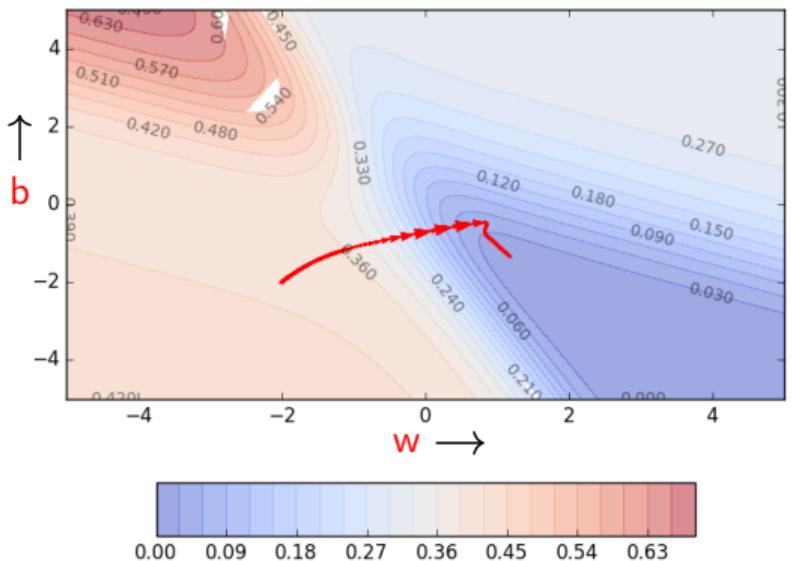
Gradient descent on the error surface



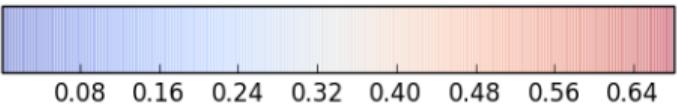
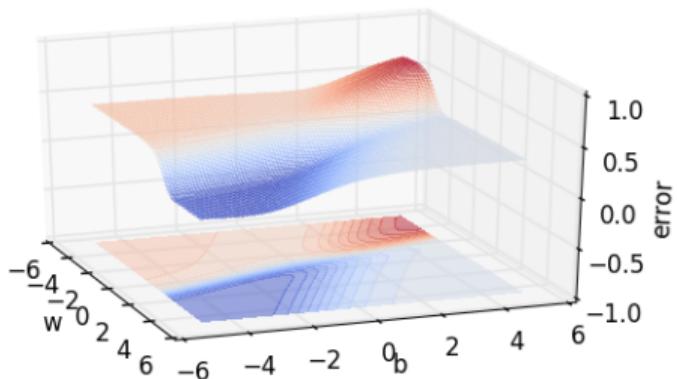


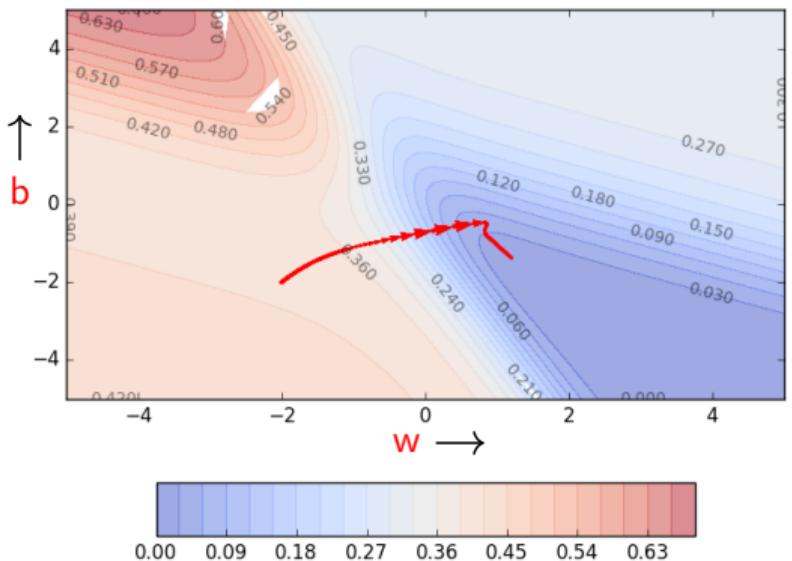
Gradient descent on the error surface



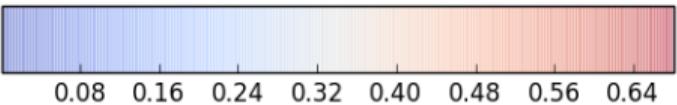
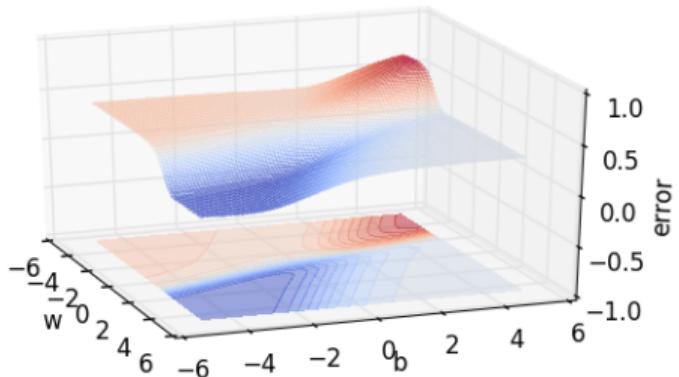


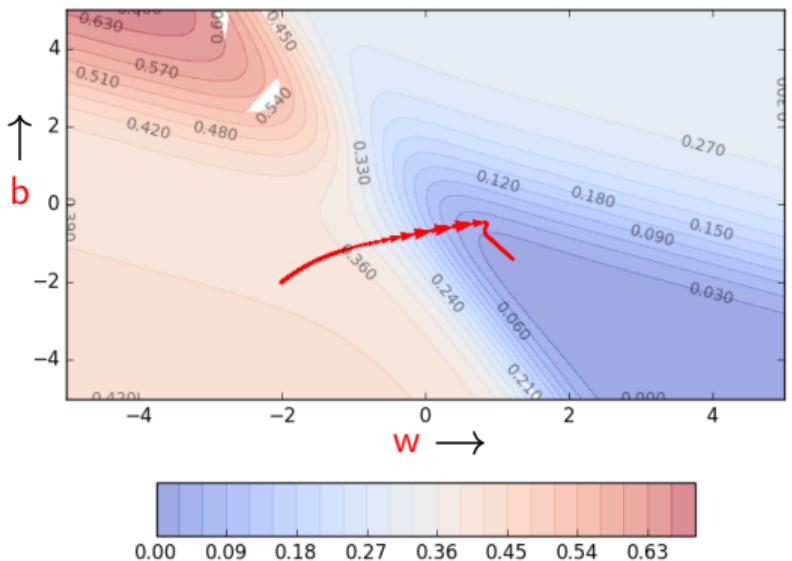
Gradient descent on the error surface



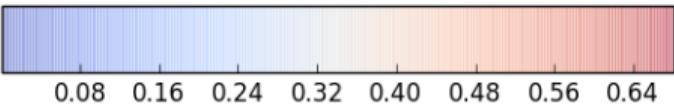
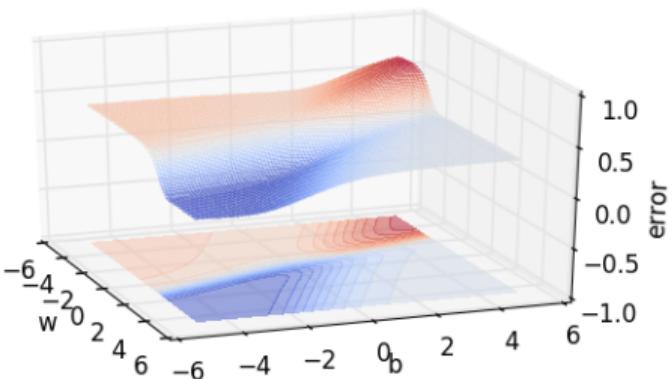


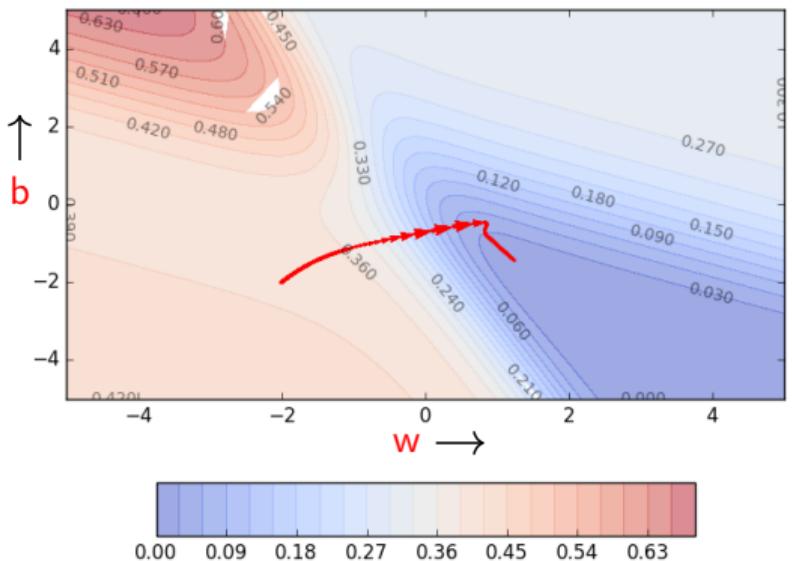
Gradient descent on the error surface



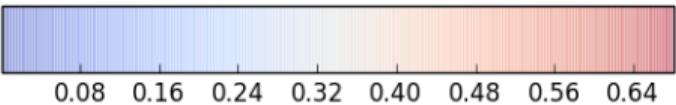
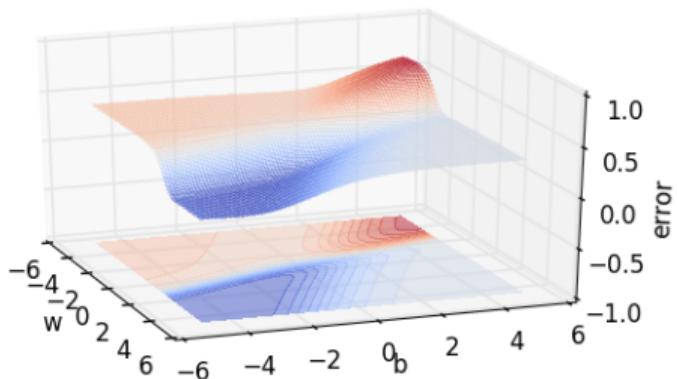


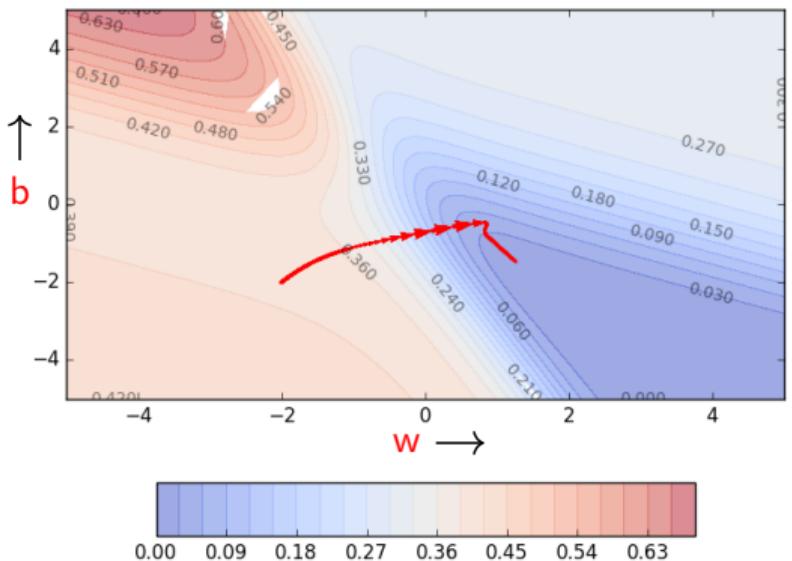
Gradient descent on the error surface



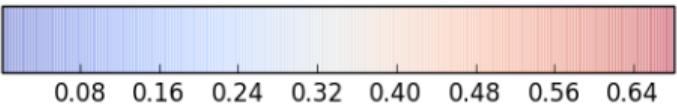
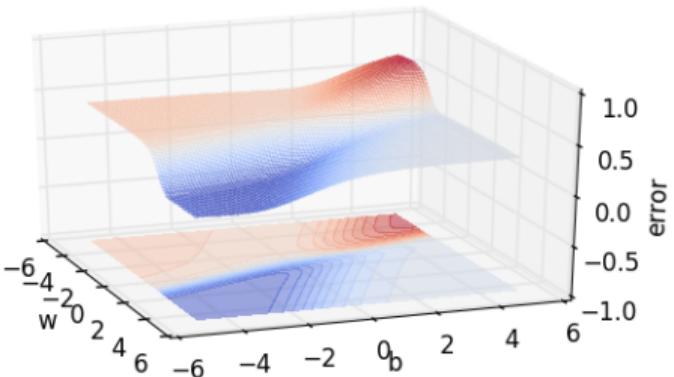


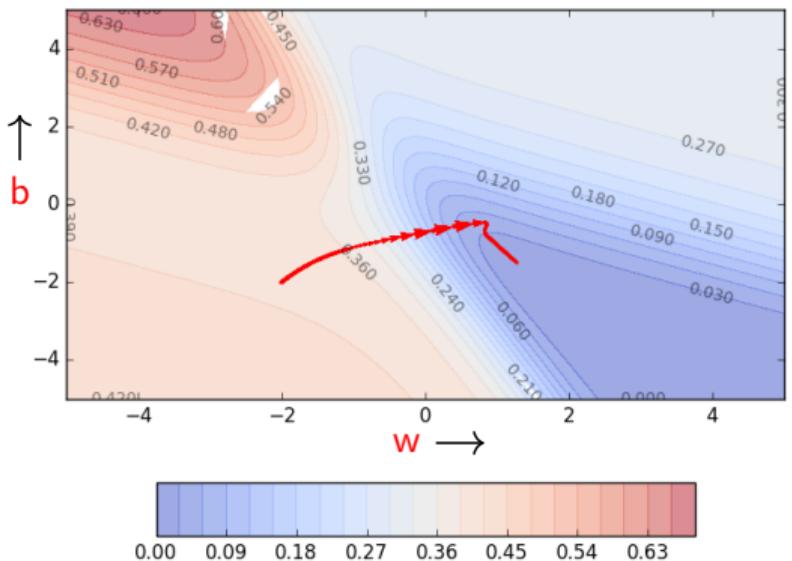
Gradient descent on the error surface



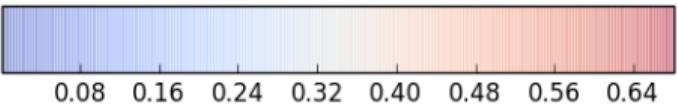
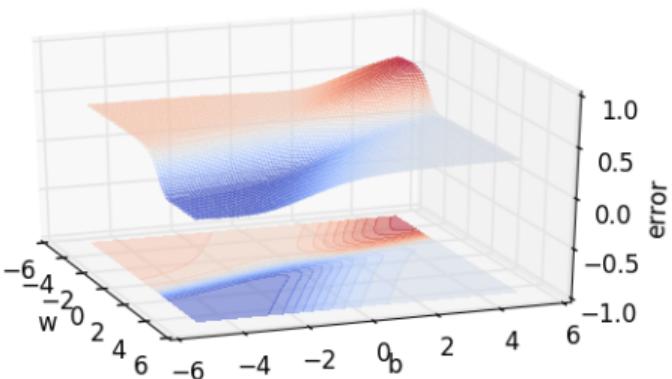


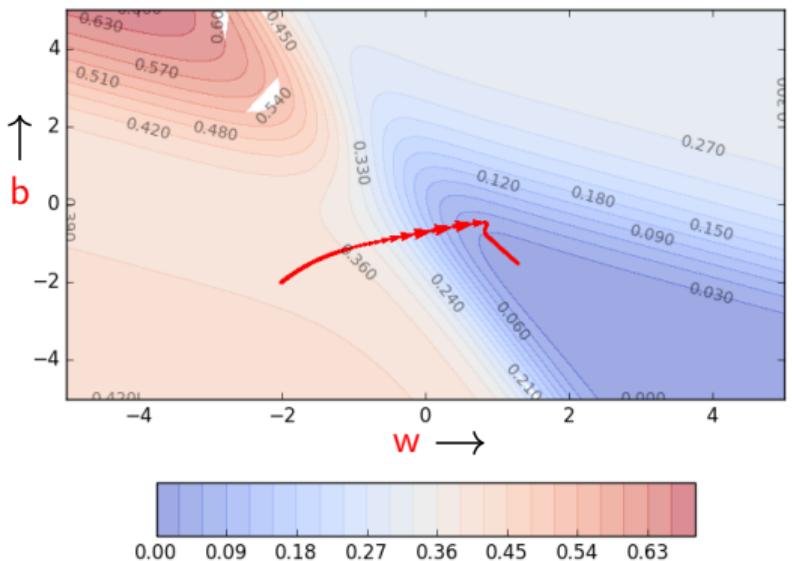
Gradient descent on the error surface



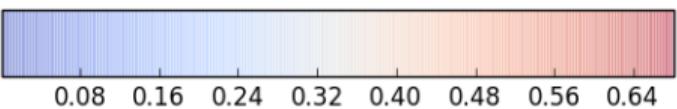
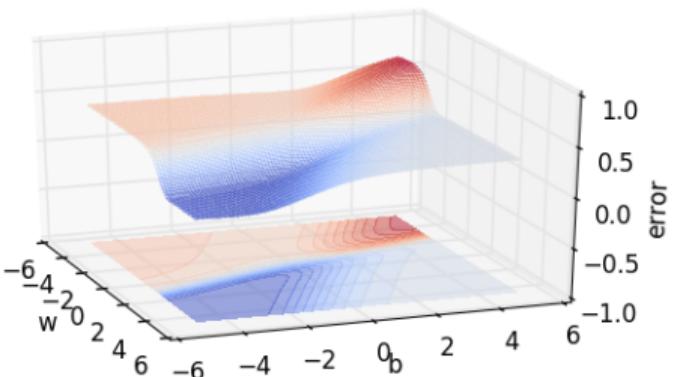


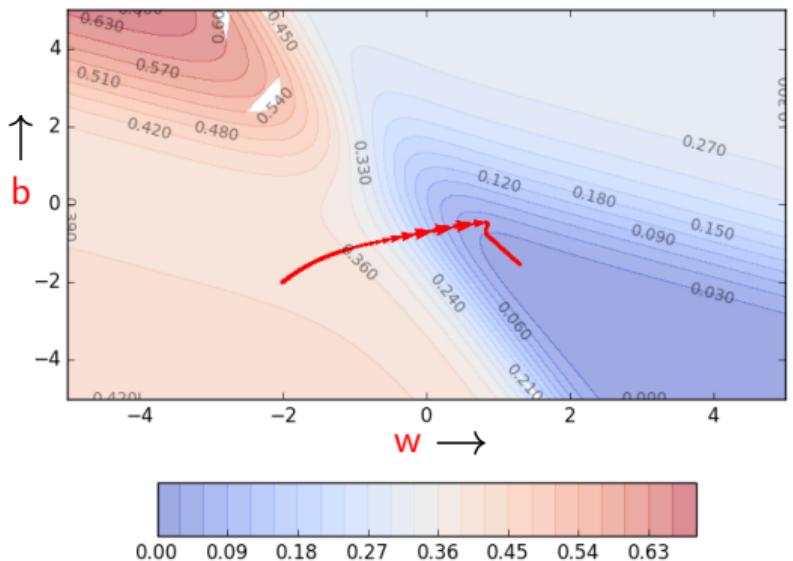
Gradient descent on the error surface



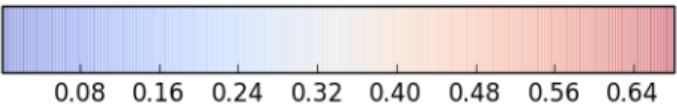
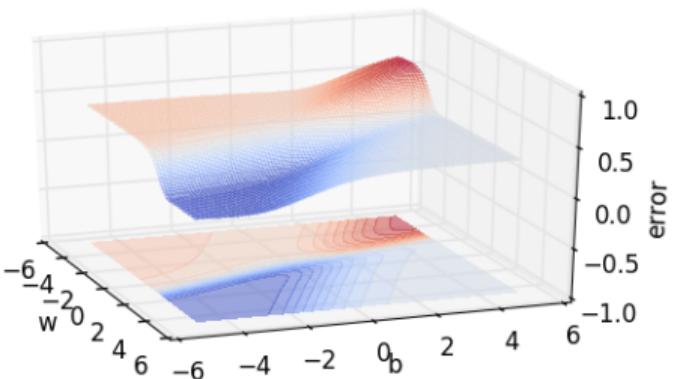


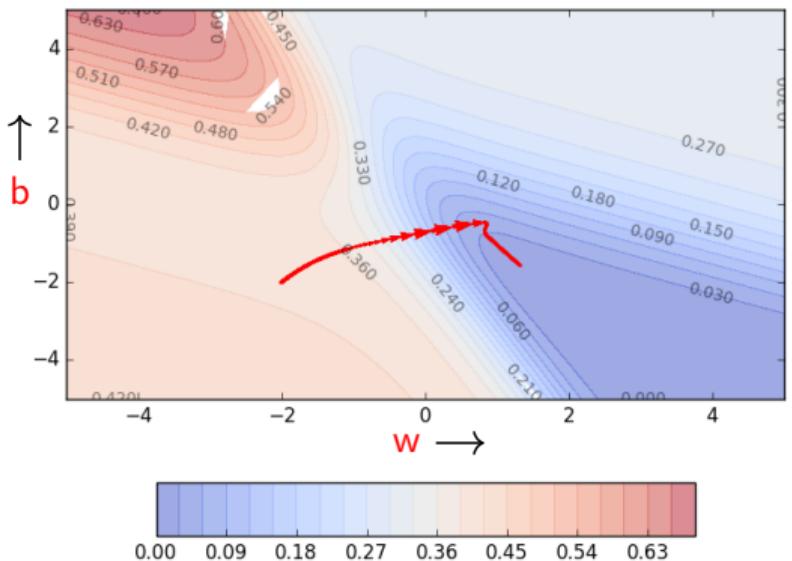
Gradient descent on the error surface



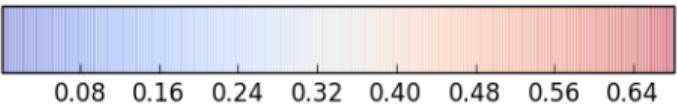
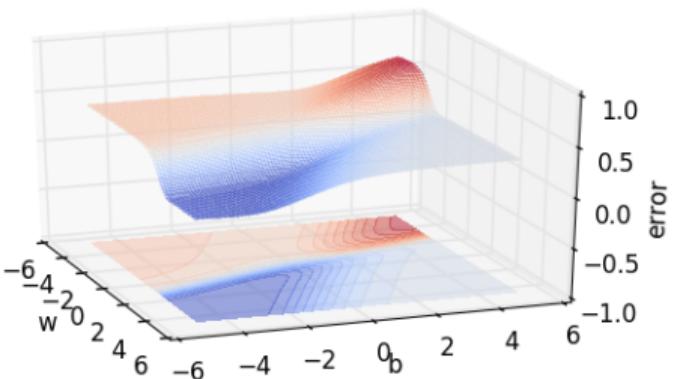


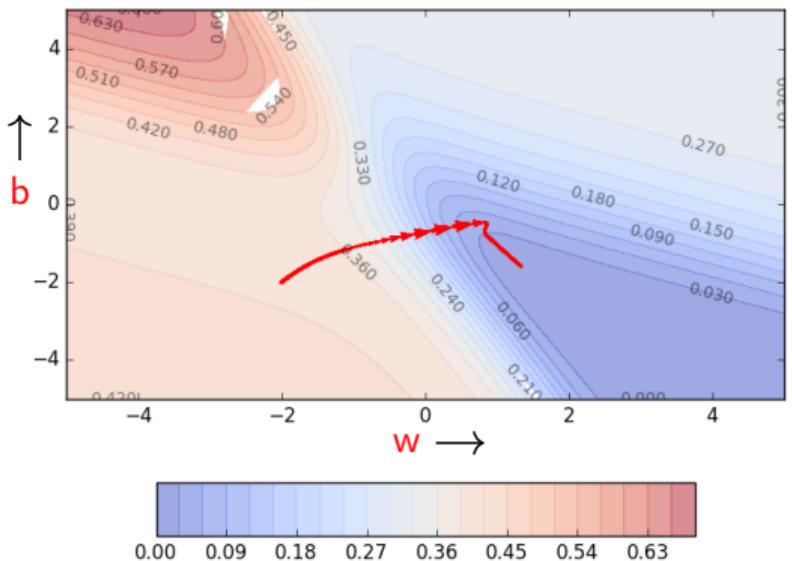
Gradient descent on the error surface



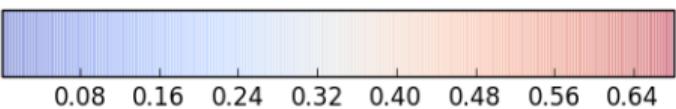
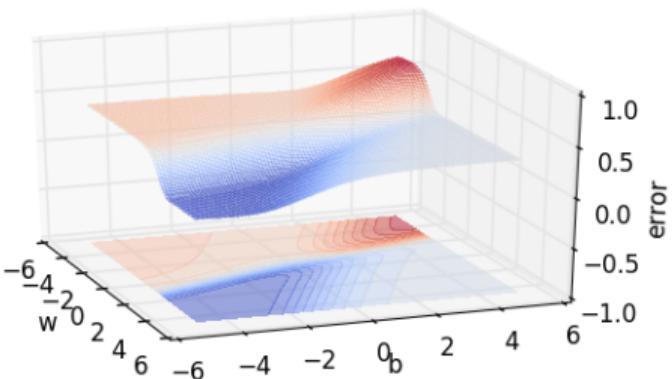


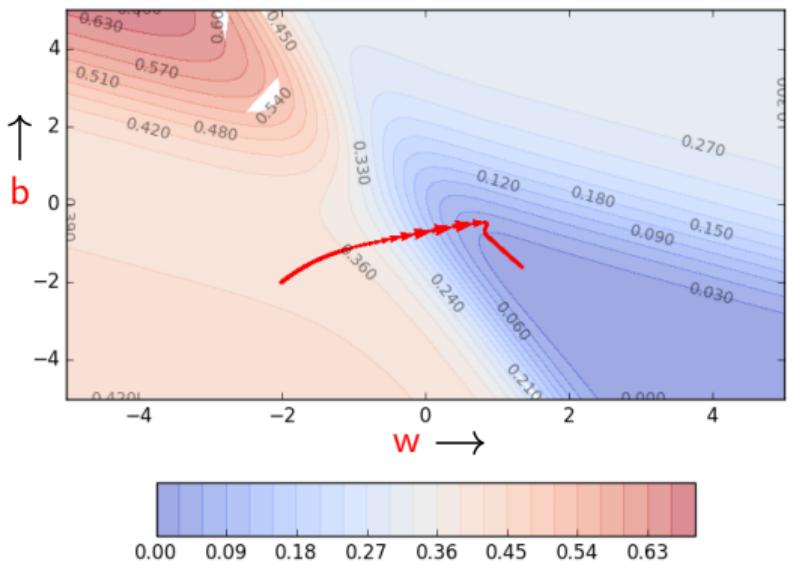
Gradient descent on the error surface



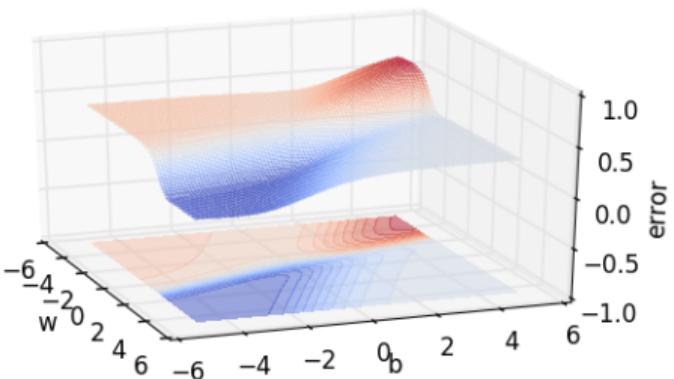


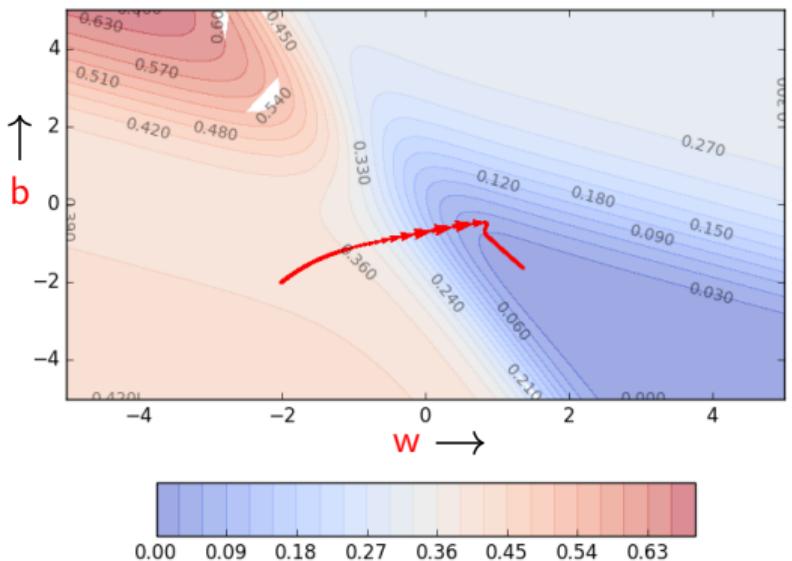
Gradient descent on the error surface



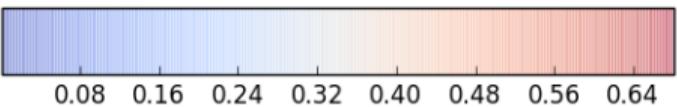
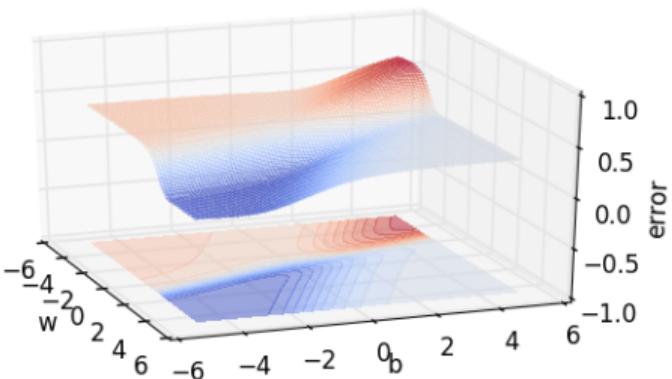


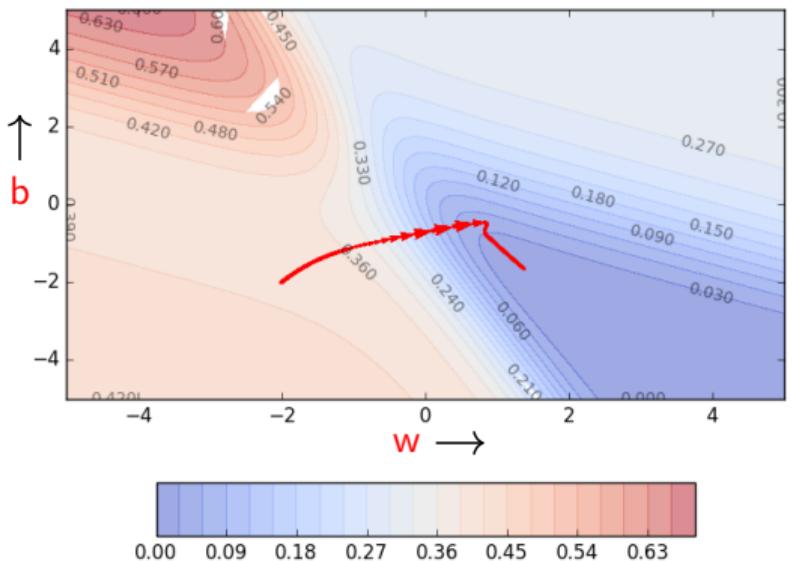
Gradient descent on the error surface



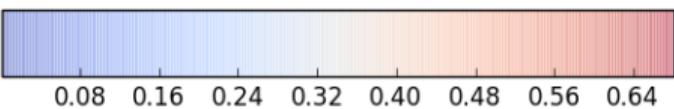
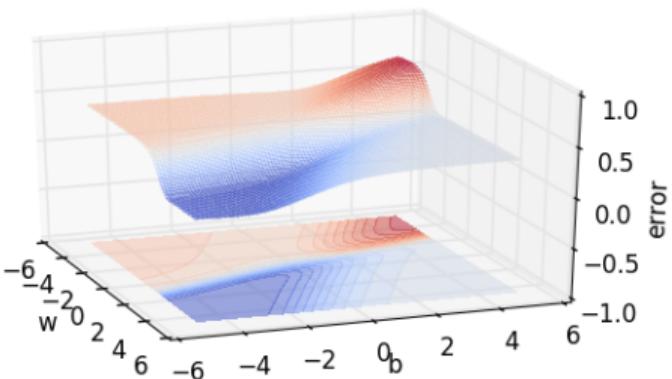


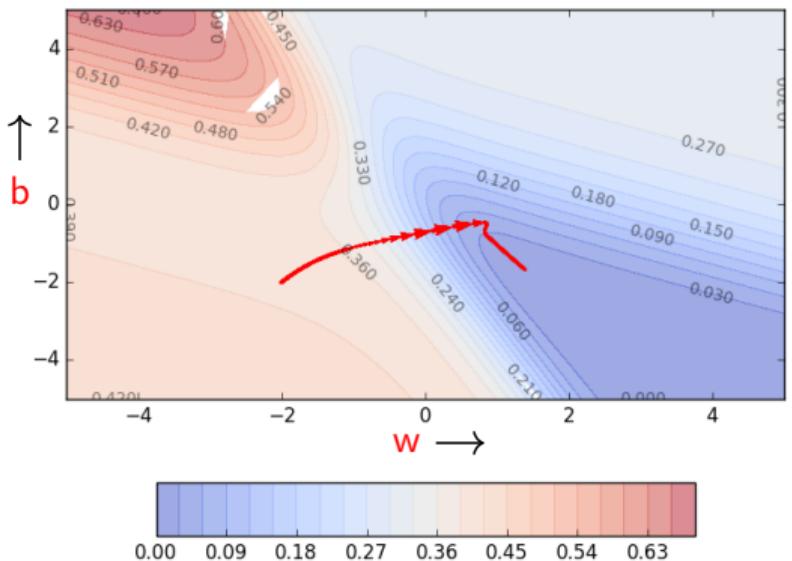
Gradient descent on the error surface



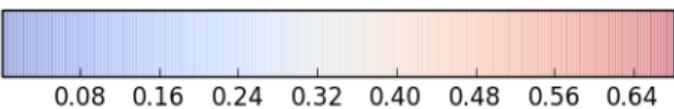
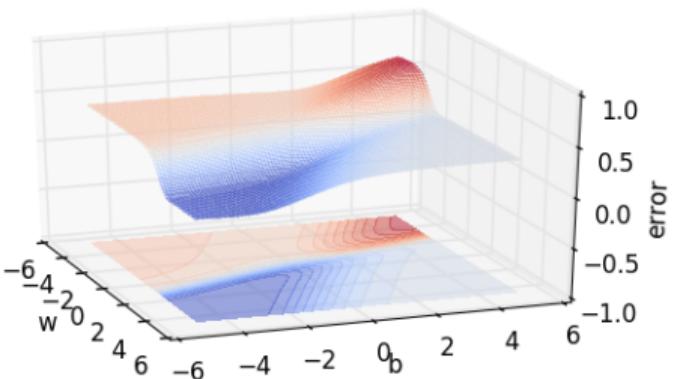


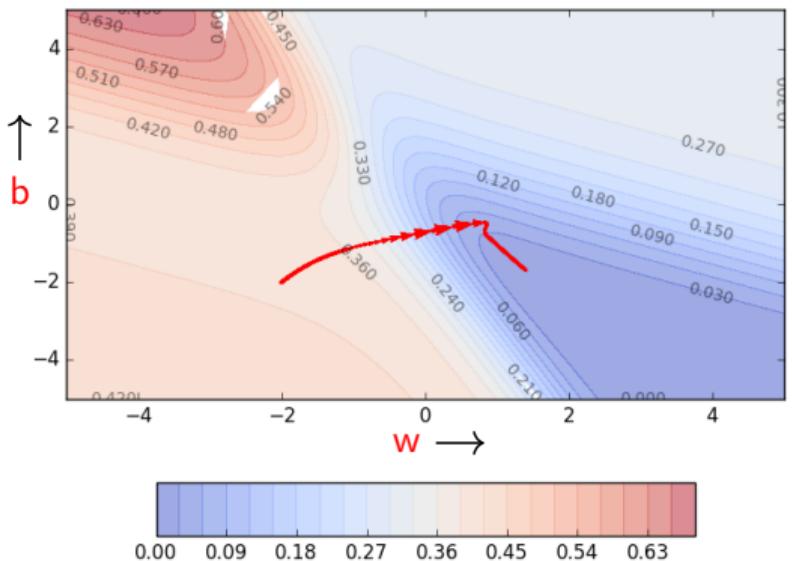
Gradient descent on the error surface



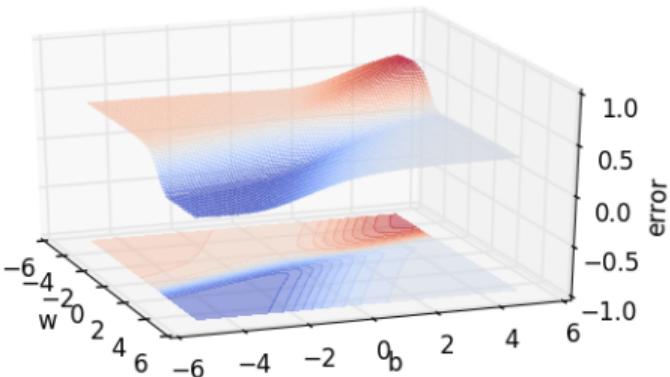


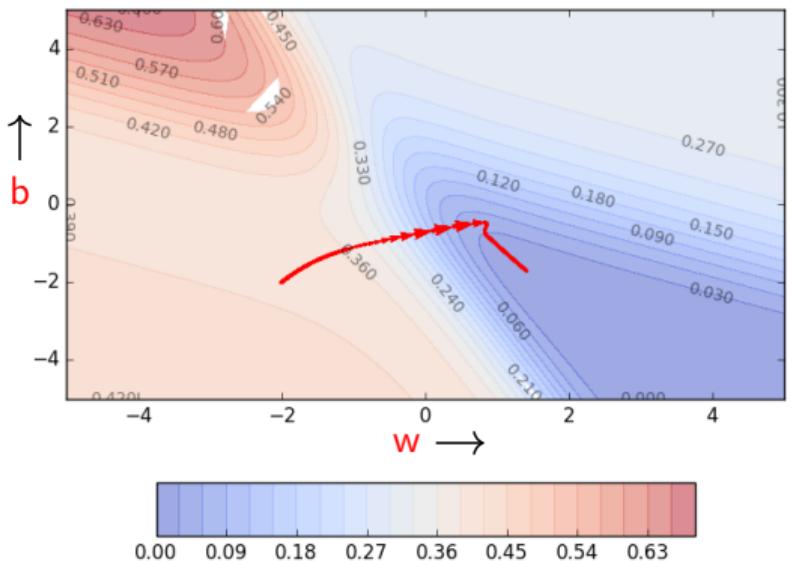
Gradient descent on the error surface



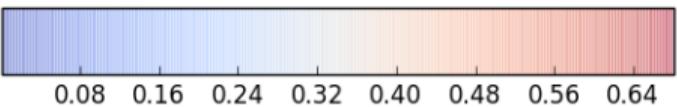
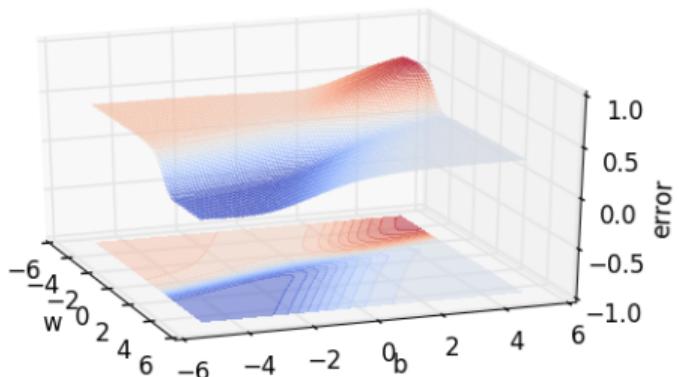


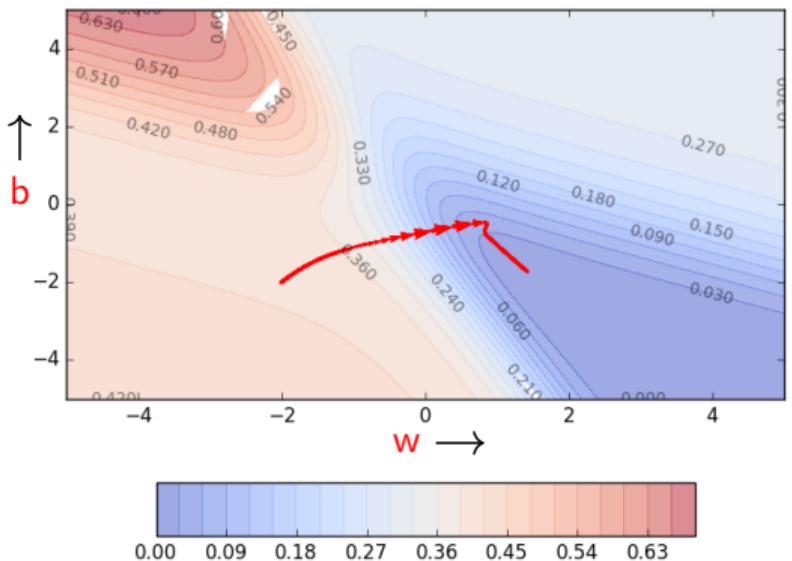
Gradient descent on the error surface



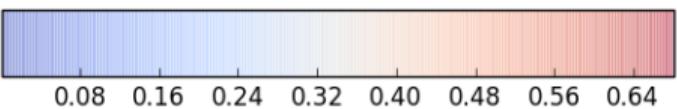
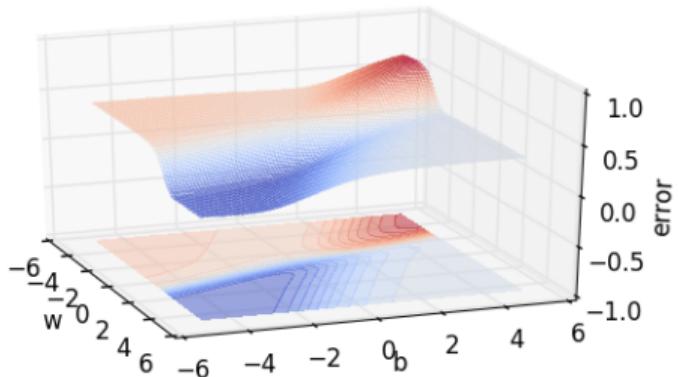


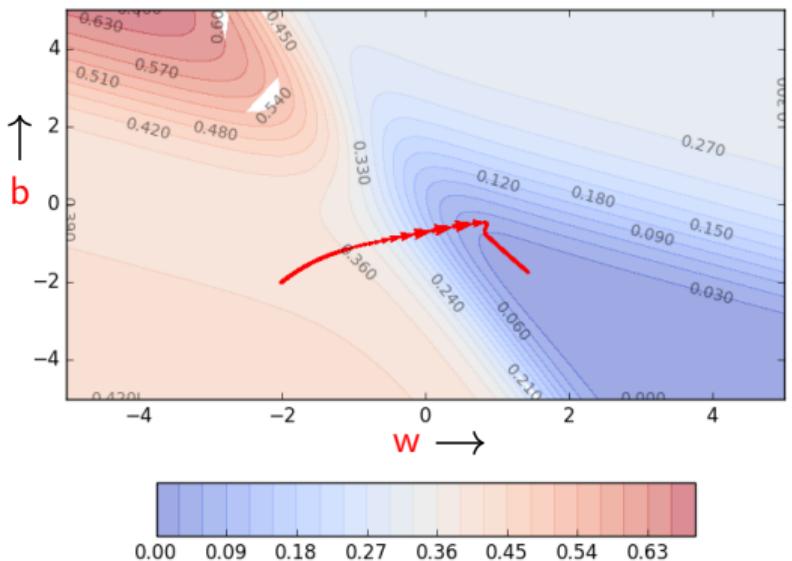
Gradient descent on the error surface



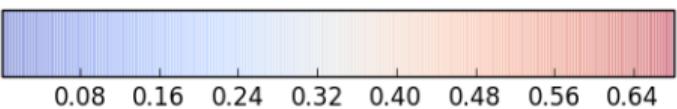
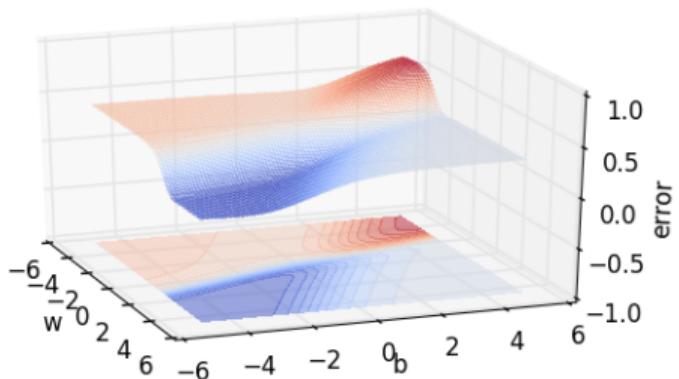


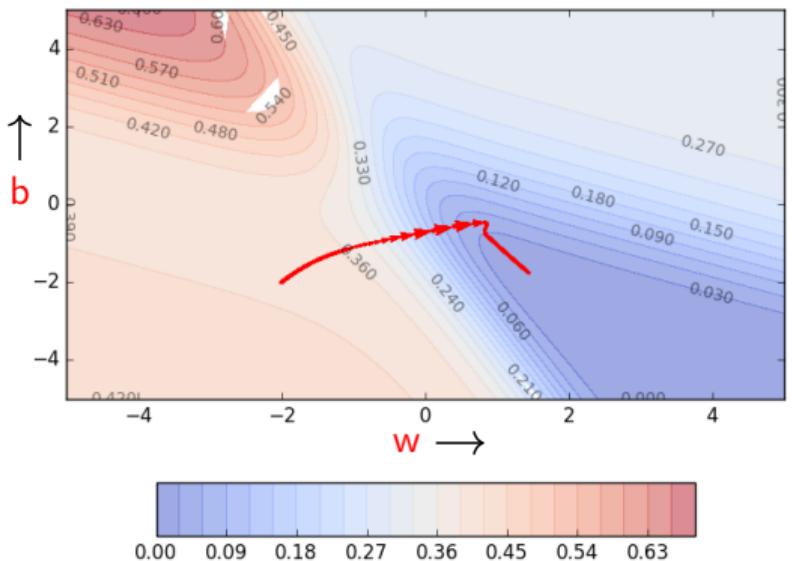
Gradient descent on the error surface



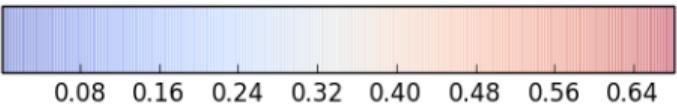
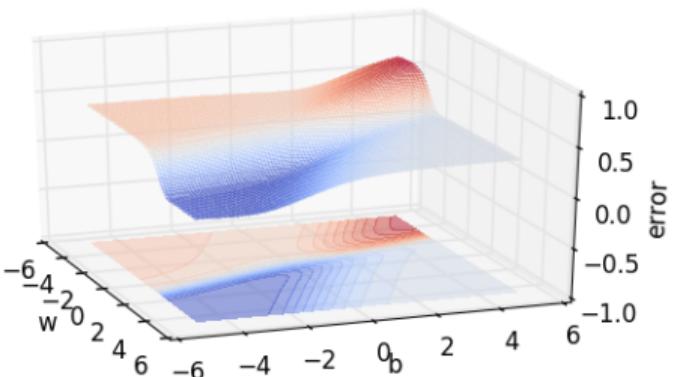


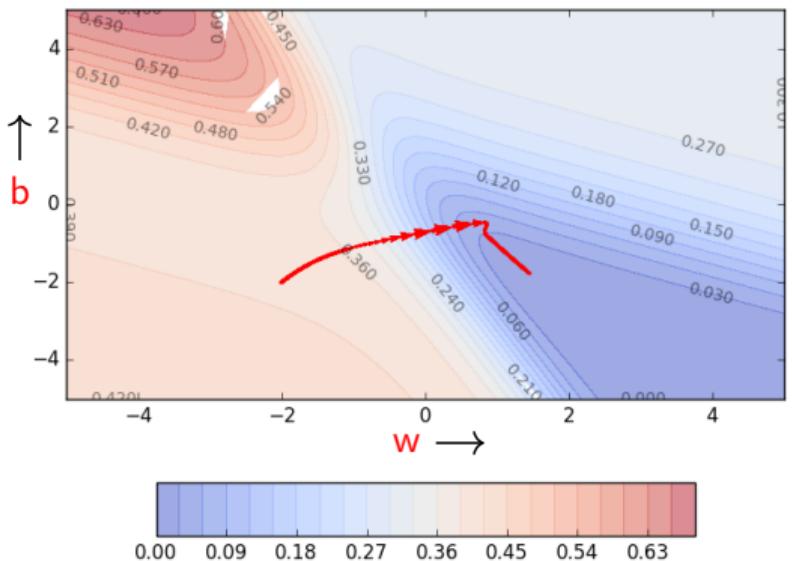
Gradient descent on the error surface



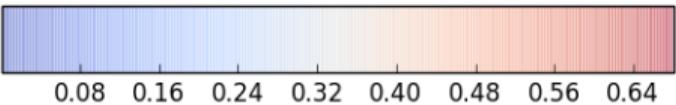
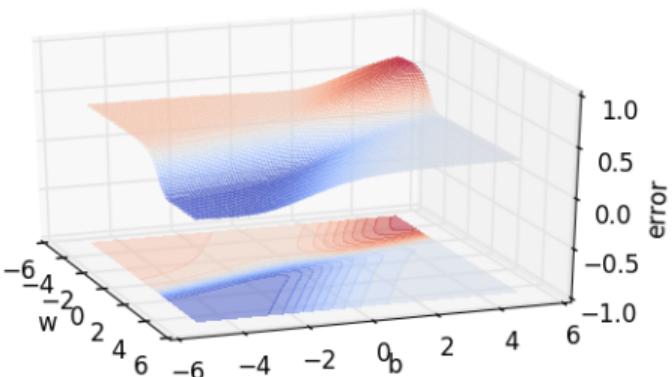


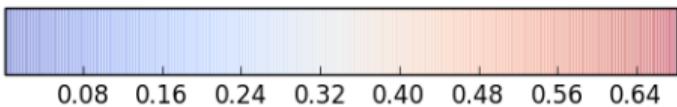
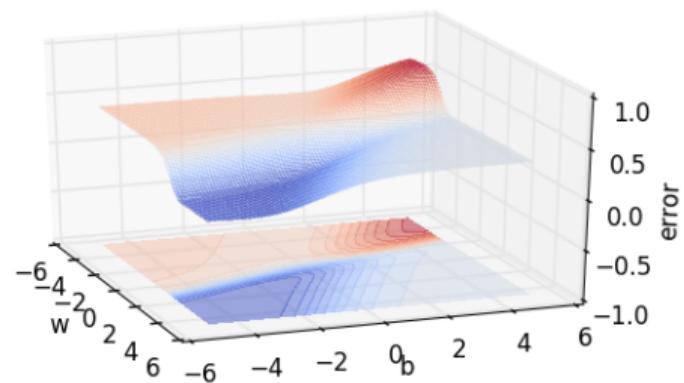
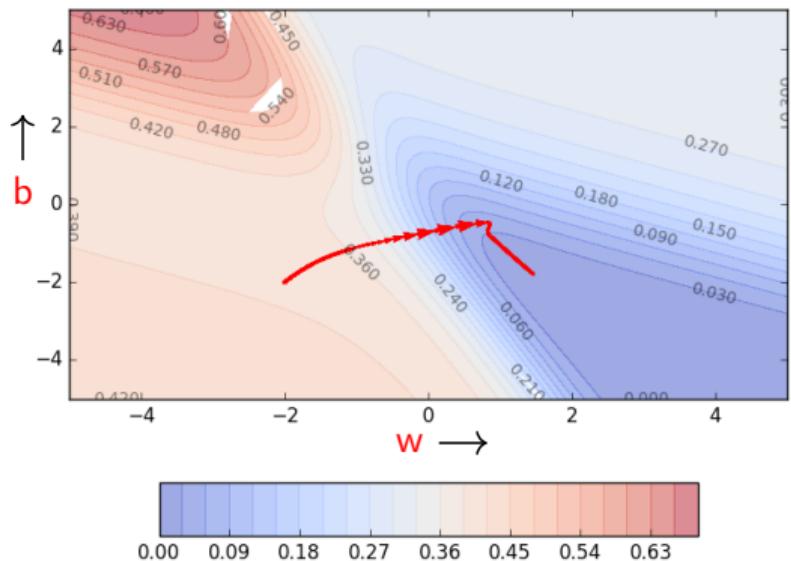
Gradient descent on the error surface





Gradient descent on the error surface





基于动量的梯度下降 (Momentum based Gradient Descent)



梯度下降算法

Algorithm 1: gradient_descent()

```
t ← 0;  
max_iterations ← 1000;  
while t < max_iterations do  
    |  $w_{t+1} \leftarrow w_t - \eta \nabla w_t;$   
    |  $b_{t+1} \leftarrow b_t - \eta \nabla b_t;$   
end
```



梯度下降算法

Algorithm 2: gradient_descent()

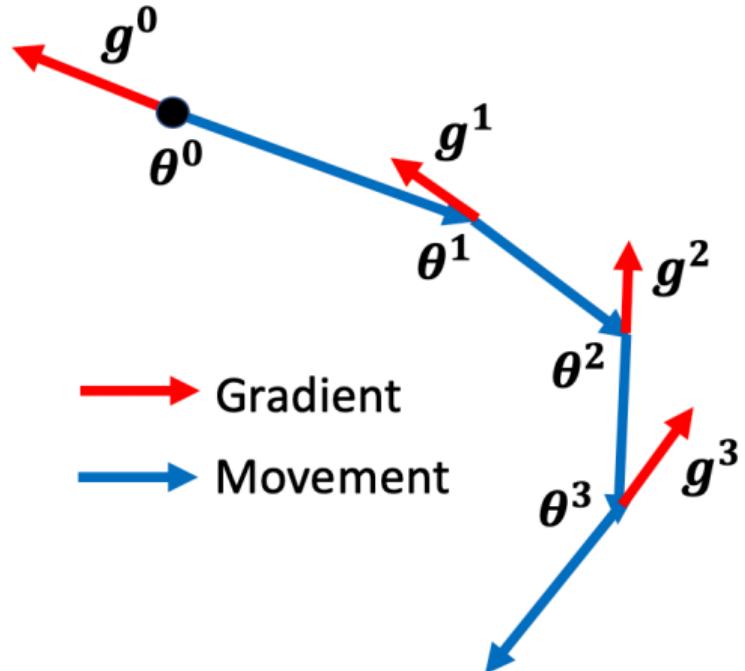
```
t ← 0;  
max_iterations ← 1000;  
while t < max_iterations do  
    |  $\theta_{t+1} \leftarrow \theta_t - \eta \nabla \theta_t$ ;  
end
```



梯度下降算法

Algorithm 3: gradient_descent()

```
t ← 0;  
max_iterations ← 1000;  
while t < max_iterations do  
    |  $\theta_{t+1} \leftarrow \theta_t - \eta \nabla \theta_t$ ;  
end
```





存在的问题

- 在遍历时，在比较平坦的区域，花费较多的时间



存在的问题

- 在遍历时，在比较平坦的区域，花费较多的时间
- 在梯度为零的点（局部最小值点或鞍点），参数不变化



存在的问题

- 在遍历时，在比较平坦的区域，花费较多的时间
- 在梯度为零的点（局部最小值点或鞍点），参数不变化
- 有没有更高效的办法？



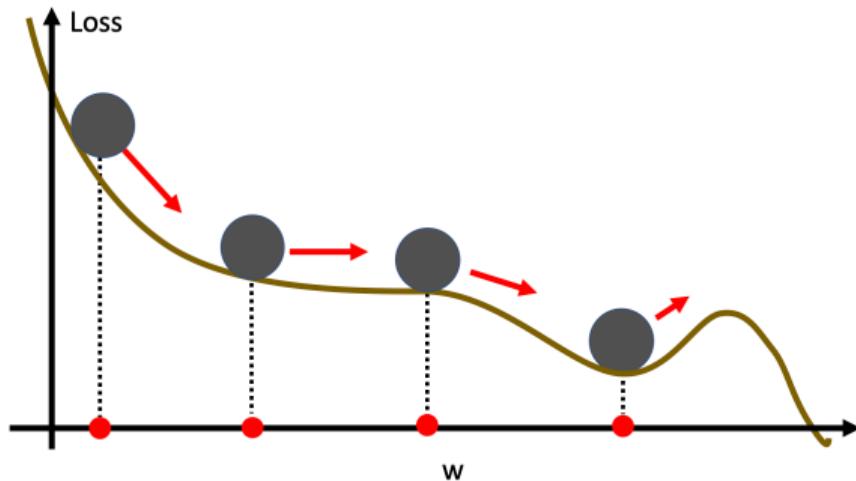
存在的问题

- 在遍历时，在比较平坦的区域，花费较多的时间
- 在梯度为零的点（局部最小值点或鞍点），参数不变化
- 有没有更高效的办法？
- 基于动量的梯度下降 ‘Momentum based gradient descent’



存在的问题

- 在遍历时，在比较平坦的区域，花费较多的时间
- 在梯度为零的点（局部最小值点或鞍点），参数不变化
- 有没有更高效的办法？
- 基于动量的梯度下降 ‘Momentum based gradient descent’





更新规则

$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$



更新规则

$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$

- 除了考虑当前更新外，还考虑上一个时刻的更新

更新规则

$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$

- 除了考虑当前更新外，还考虑上一个时刻的更新

当前的 update，不仅基于当前梯度，还基于先前的 update

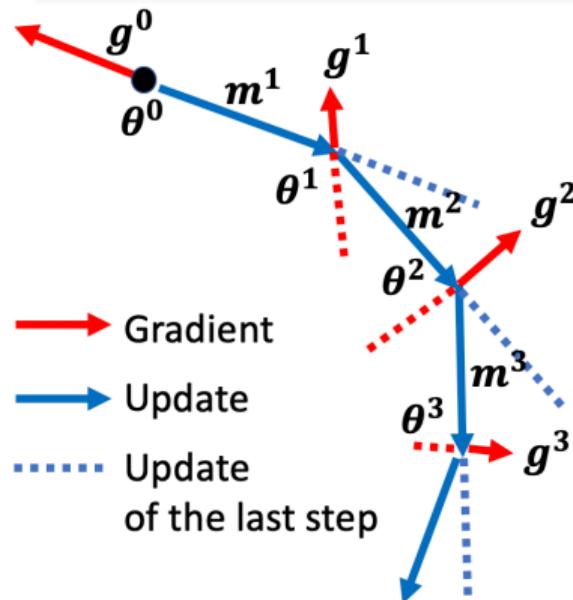
更新规则

$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$

- 除了考虑当前更新外，还考虑上一个时刻的更新

当前的 update，不仅基于当前梯度，还基于先前的 update





$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$

$$update_0 = 0$$



$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$

$$update_0 = 0$$

$$update_1 = \gamma \cdot update_0 - \eta \nabla w_1 = -\eta \nabla w_1$$



$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$

$$update_0 = 0$$

$$update_1 = \gamma \cdot update_0 - \eta \nabla w_1 = -\eta \nabla w_1$$

$$update_2 = \gamma \cdot update_1 - \eta \nabla w_2 = -\gamma \cdot \eta \nabla w_1 - \eta \nabla w_2$$

$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$

$$update_0 = 0$$

$$update_1 = \gamma \cdot update_0 - \eta \nabla w_1 = -\eta \nabla w_1$$

$$update_2 = \gamma \cdot update_1 - \eta \nabla w_2 = -\gamma \cdot \eta \nabla w_1 - \eta \nabla w_2$$

$$update_3 = \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 - \eta \nabla w_2) - \eta \nabla w_3$$

$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$

$$update_0 = 0$$

$$update_1 = \gamma \cdot update_0 - \eta \nabla w_1 = -\eta \nabla w_1$$

$$update_2 = \gamma \cdot update_1 - \eta \nabla w_2 = -\gamma \cdot \eta \nabla w_1 - \eta \nabla w_2$$

$$update_3 = \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 - \eta \nabla w_2) - \eta \nabla w_3$$

$$= \gamma \cdot update_2 + \eta \nabla w_3 = -\gamma^2 \cdot \eta \nabla w_1 - \gamma \cdot \eta \nabla w_2 - \eta \nabla w_3$$

$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$

$$update_0 = 0$$

$$update_1 = \gamma \cdot update_0 - \eta \nabla w_1 = -\eta \nabla w_1$$

$$update_2 = \gamma \cdot update_1 - \eta \nabla w_2 = -\gamma \cdot \eta \nabla w_1 - \eta \nabla w_2$$

$$update_3 = \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 - \eta \nabla w_2) - \eta \nabla w_3$$

$$= \gamma \cdot update_2 + \eta \nabla w_3 = -\gamma^2 \cdot \eta \nabla w_1 - \gamma \cdot \eta \nabla w_2 - \eta \nabla w_3$$

$$update_4 = \gamma \cdot update_3 + \eta \nabla w_4 = -\gamma^3 \cdot \eta \nabla w_1 - \gamma^2 \cdot \eta \nabla w_2 - \gamma \cdot \eta \nabla w_3 - \eta \nabla w_4$$

$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$

$$update_0 = 0$$

$$update_1 = \gamma \cdot update_0 - \eta \nabla w_1 = -\eta \nabla w_1$$

$$update_2 = \gamma \cdot update_1 - \eta \nabla w_2 = -\gamma \cdot \eta \nabla w_1 - \eta \nabla w_2$$

$$update_3 = \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 - \eta \nabla w_2) - \eta \nabla w_3$$

$$= \gamma \cdot update_2 + \eta \nabla w_3 = -\gamma^2 \cdot \eta \nabla w_1 - \gamma \cdot \eta \nabla w_2 - \eta \nabla w_3$$

$$update_4 = \gamma \cdot update_3 + \eta \nabla w_4 = -\gamma^3 \cdot \eta \nabla w_1 - \gamma^2 \cdot \eta \nabla w_2 - \gamma \cdot \eta \nabla w_3 - \eta \nabla w_4$$

⋮

$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$$

$$w_{t+1} = w_t + update_t$$

$$update_0 = 0$$

$$update_1 = \gamma \cdot update_0 - \eta \nabla w_1 = -\eta \nabla w_1$$

$$update_2 = \gamma \cdot update_1 - \eta \nabla w_2 = -\gamma \cdot \eta \nabla w_1 - \eta \nabla w_2$$

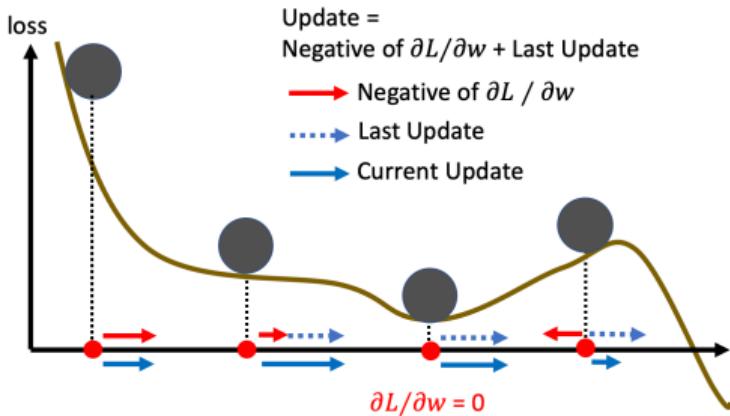
$$update_3 = \gamma \cdot update_2 + \eta \nabla w_3 = \gamma(\gamma \cdot \eta \nabla w_1 - \eta \nabla w_2) - \eta \nabla w_3$$

$$= \gamma \cdot update_2 + \eta \nabla w_3 = -\gamma^2 \cdot \eta \nabla w_1 - \gamma \cdot \eta \nabla w_2 - \eta \nabla w_3$$

$$update_4 = \gamma \cdot update_3 + \eta \nabla w_4 = -\gamma^3 \cdot \eta \nabla w_1 - \gamma^2 \cdot \eta \nabla w_2 - \gamma \cdot \eta \nabla w_3 - \eta \nabla w_4$$

⋮

$$update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t = -\gamma^{t-1} \cdot \eta \nabla w_1 - \gamma^{t-2} \cdot \eta \nabla w_1 - \dots - \eta \nabla w_t$$

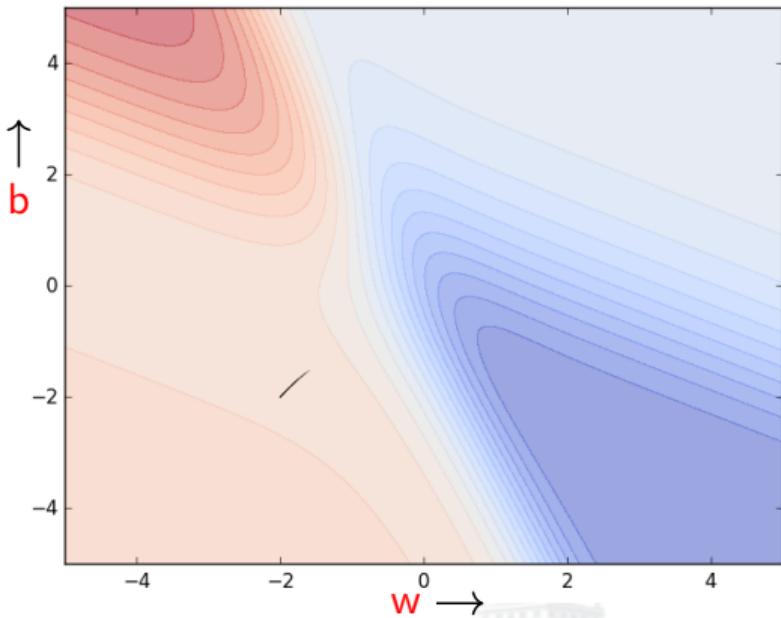




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

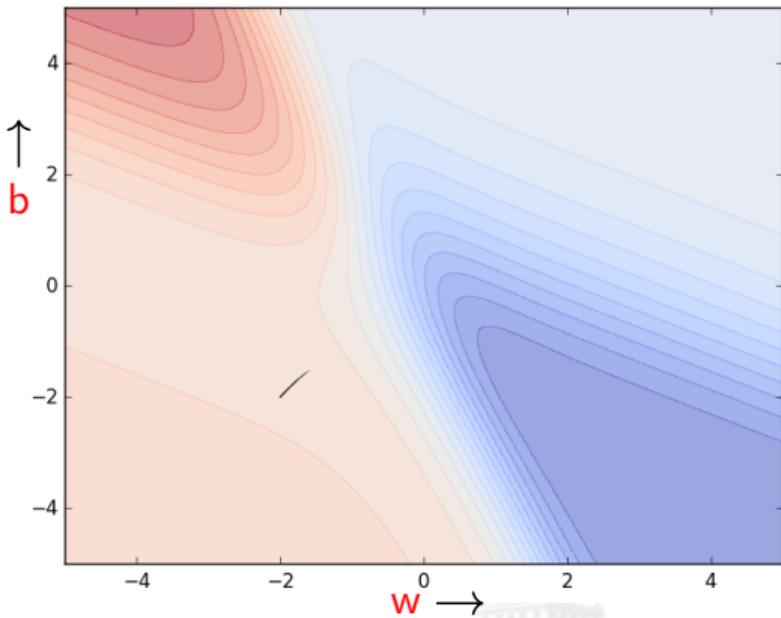




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

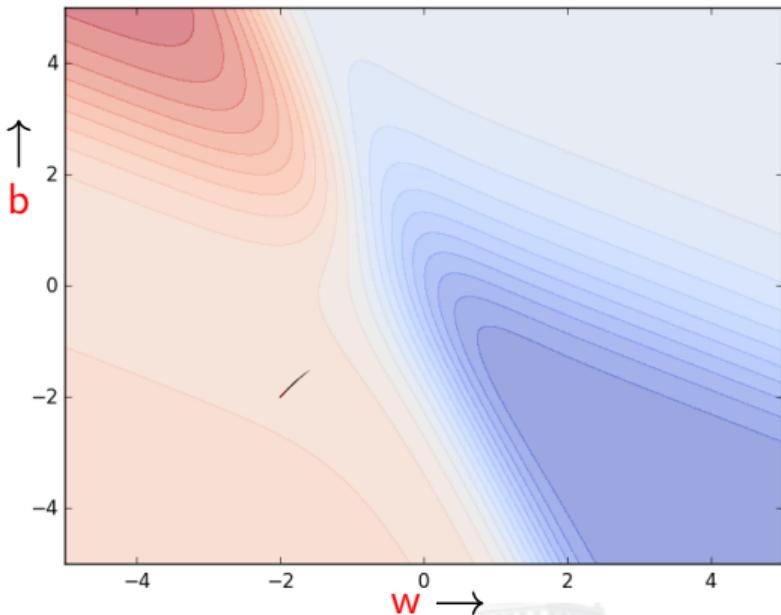




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

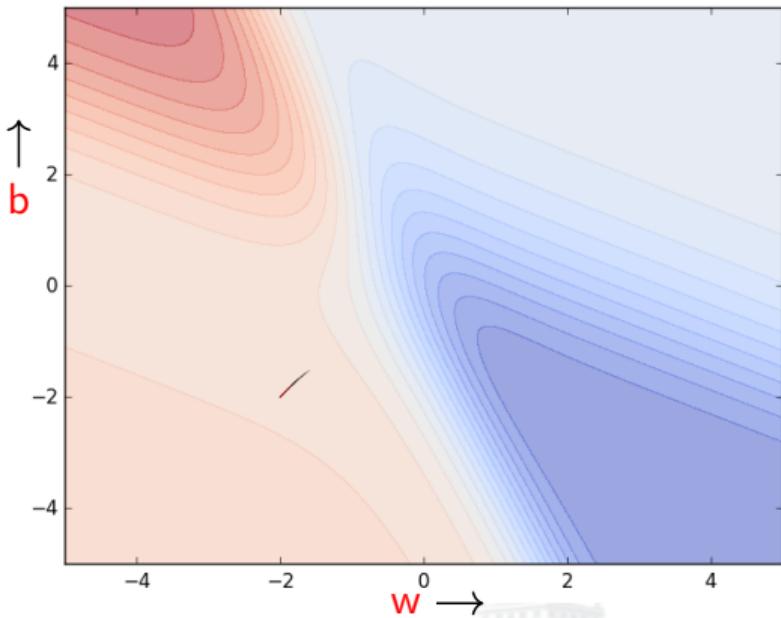




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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)

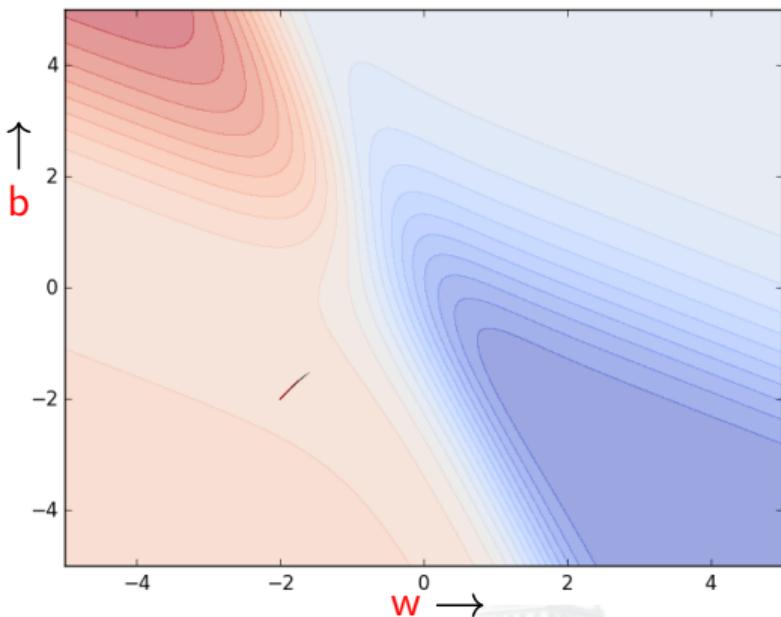
        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```



```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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)

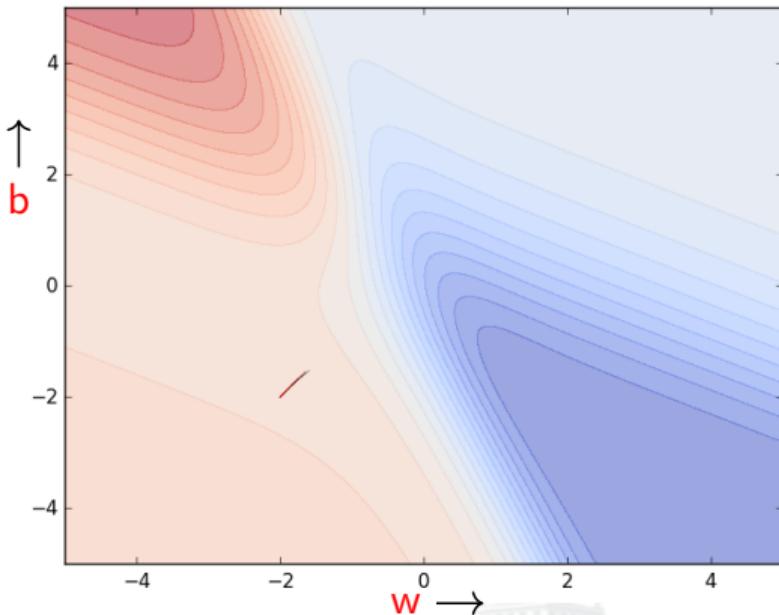
        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```



```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

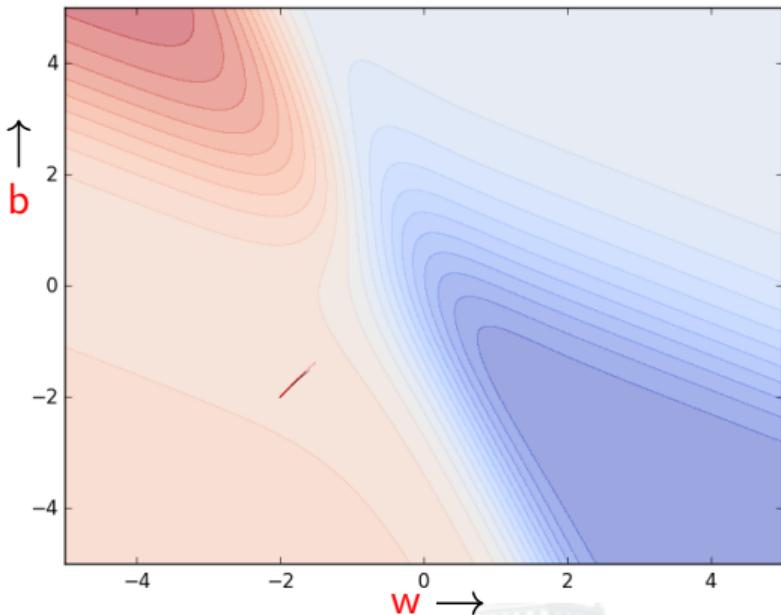




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

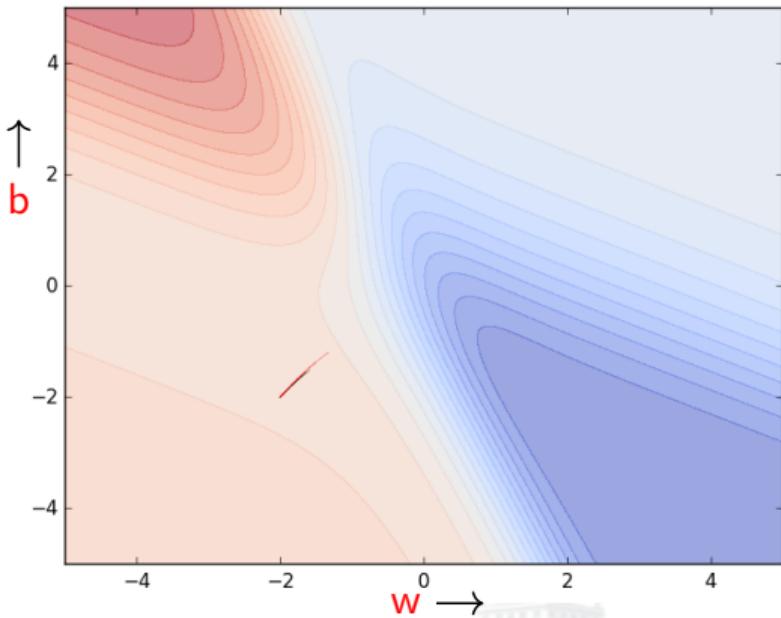




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

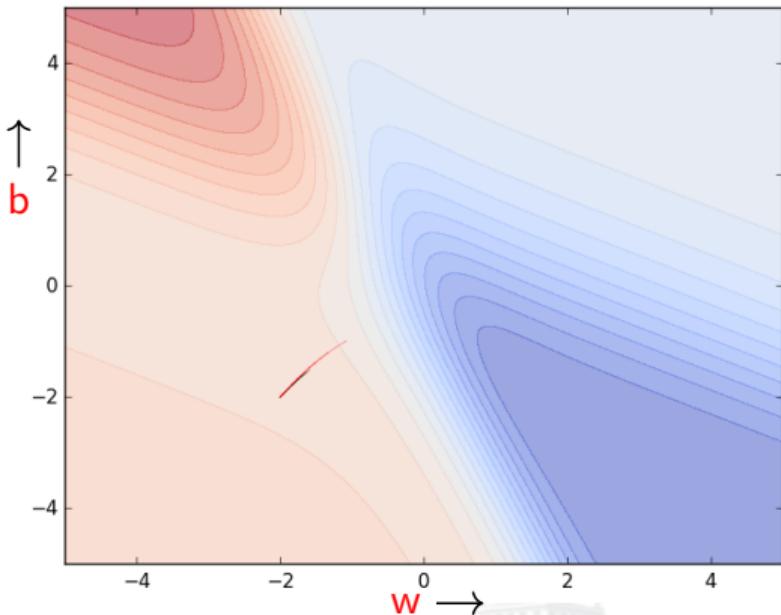




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

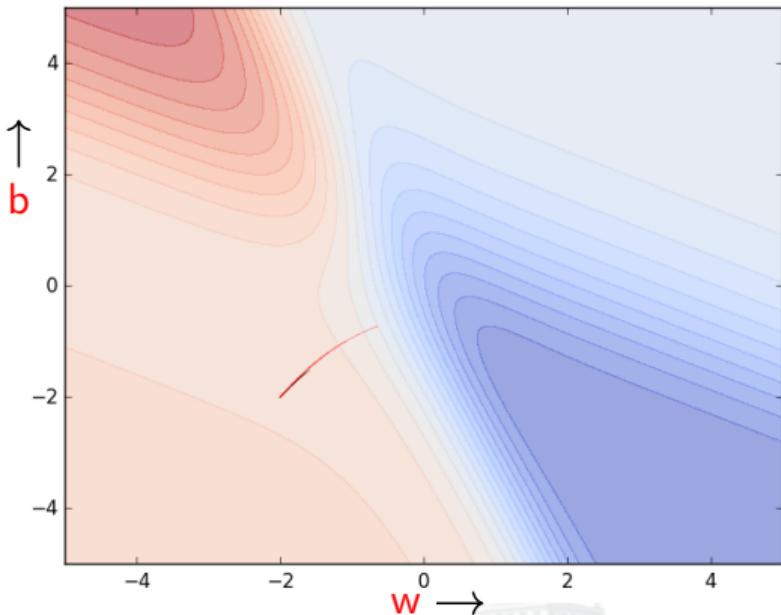




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

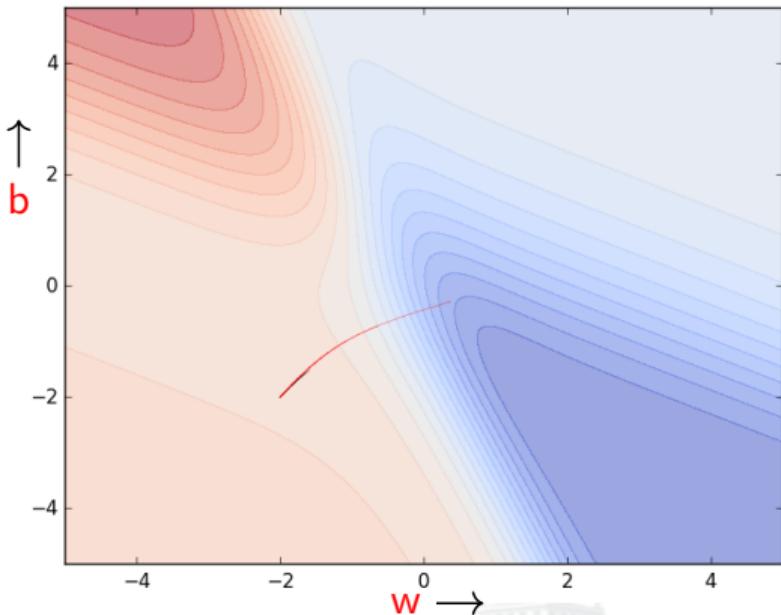




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

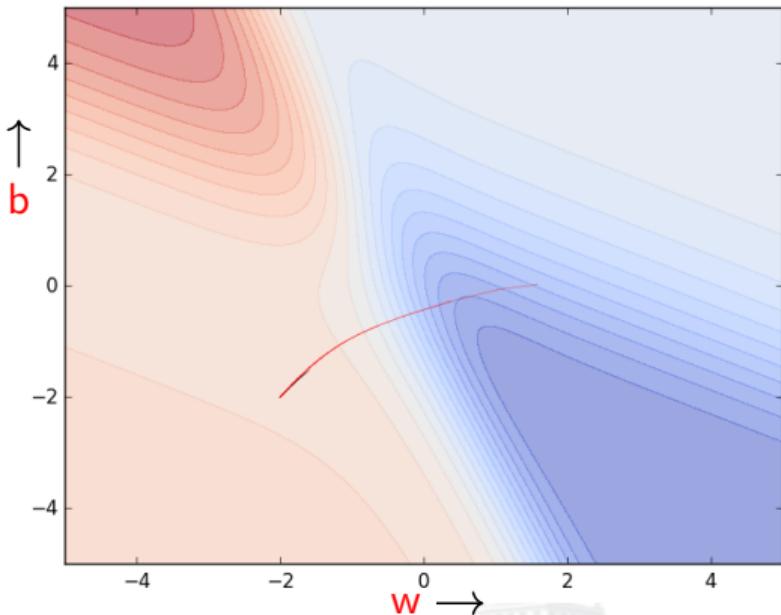




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

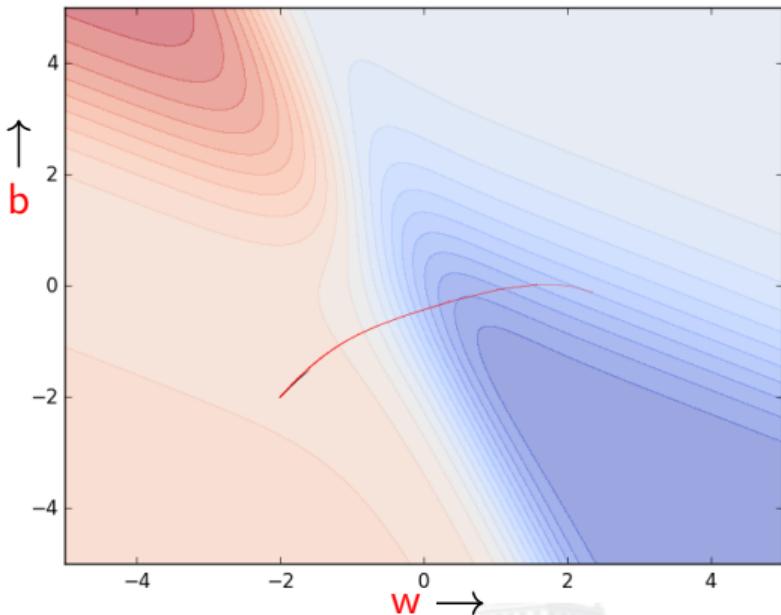




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

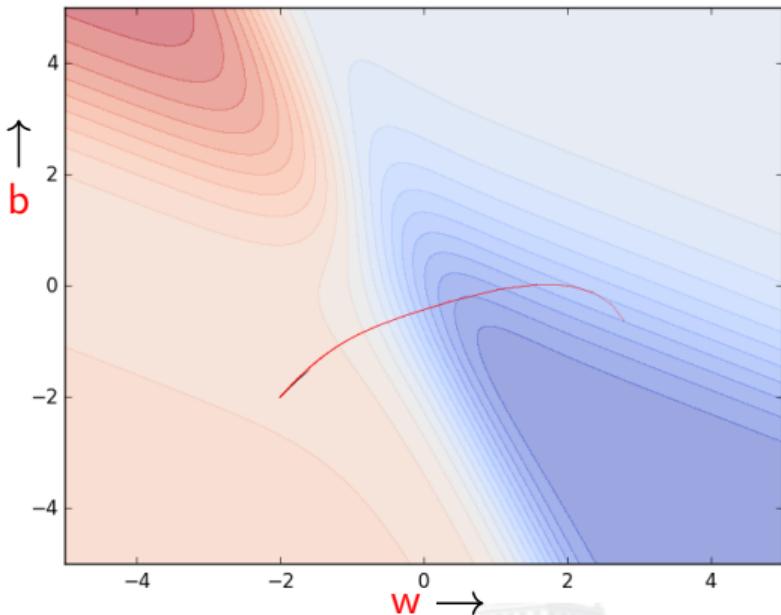




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

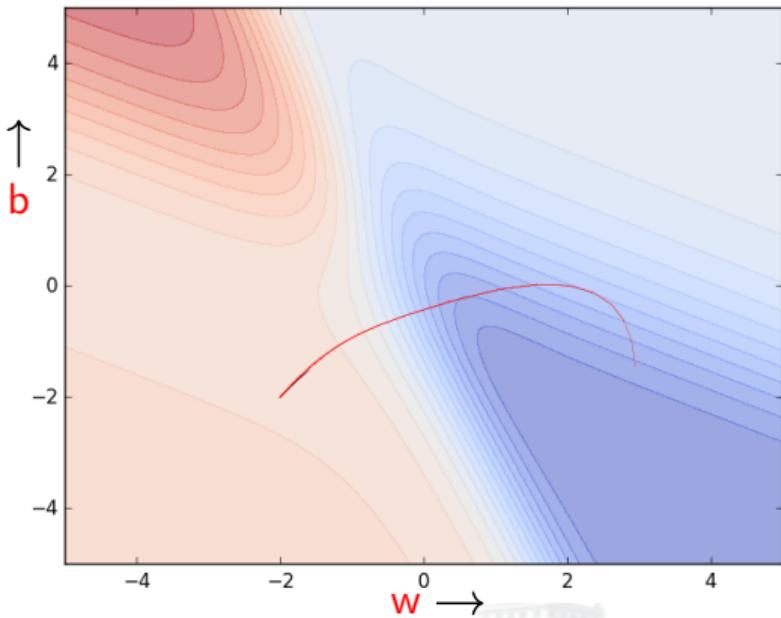




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

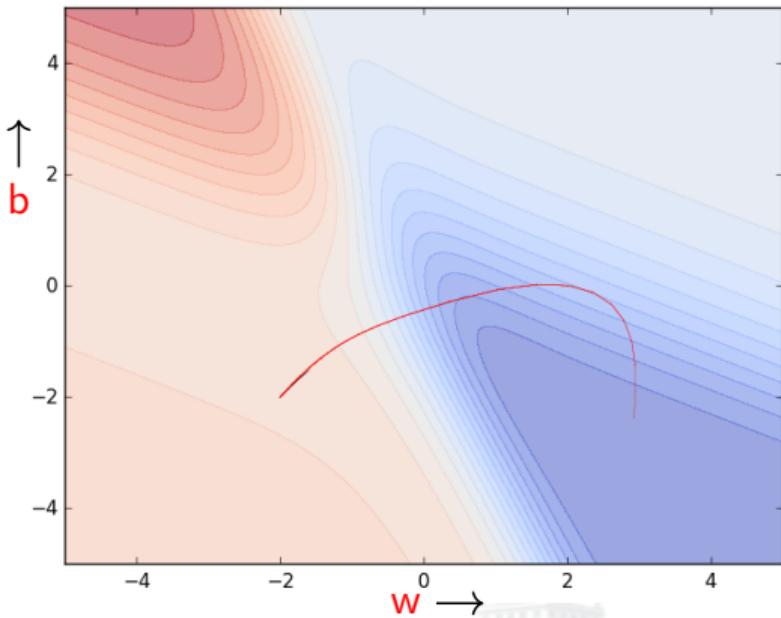




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

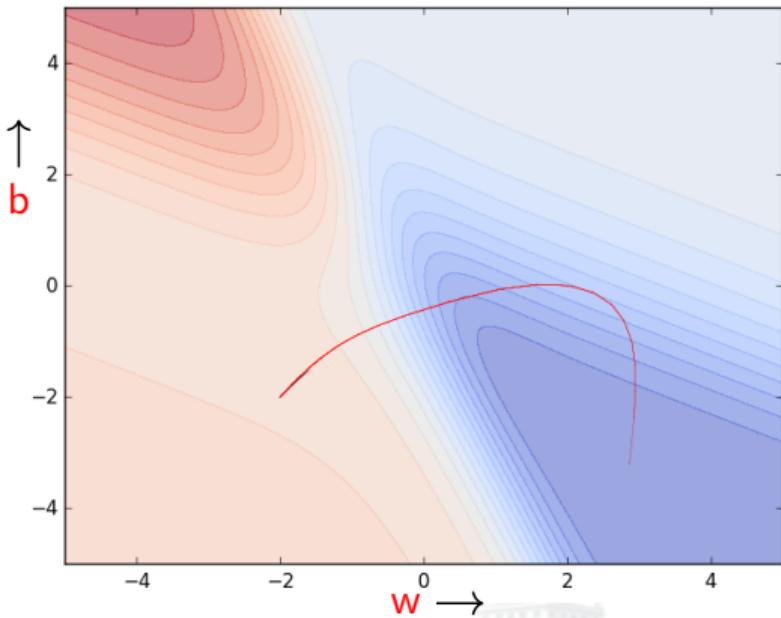




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

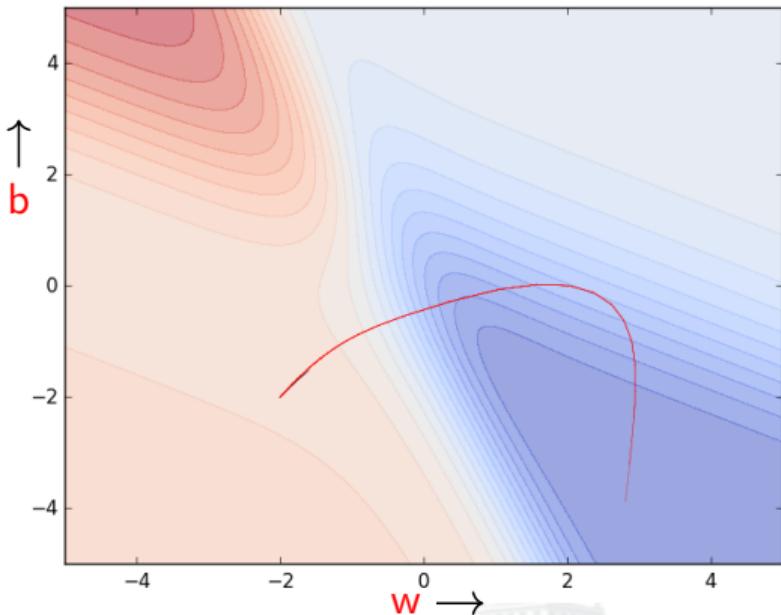




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```

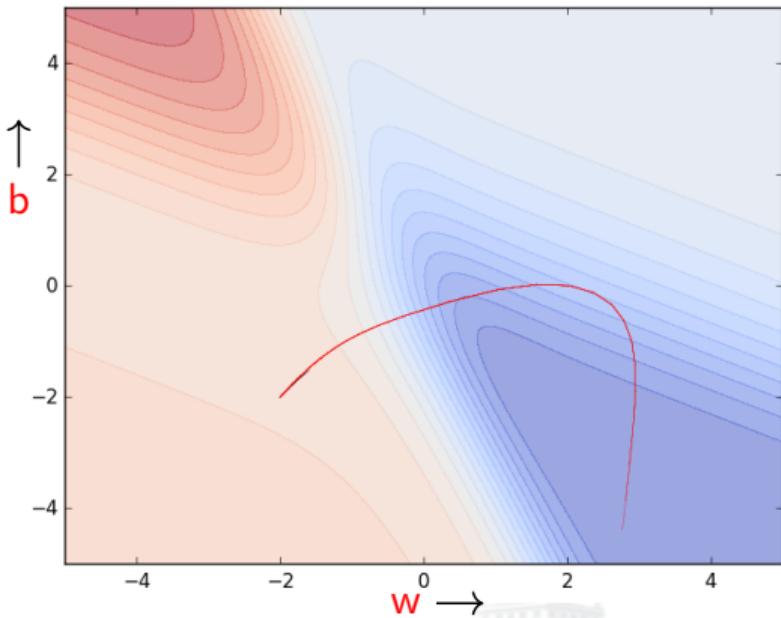




```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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)

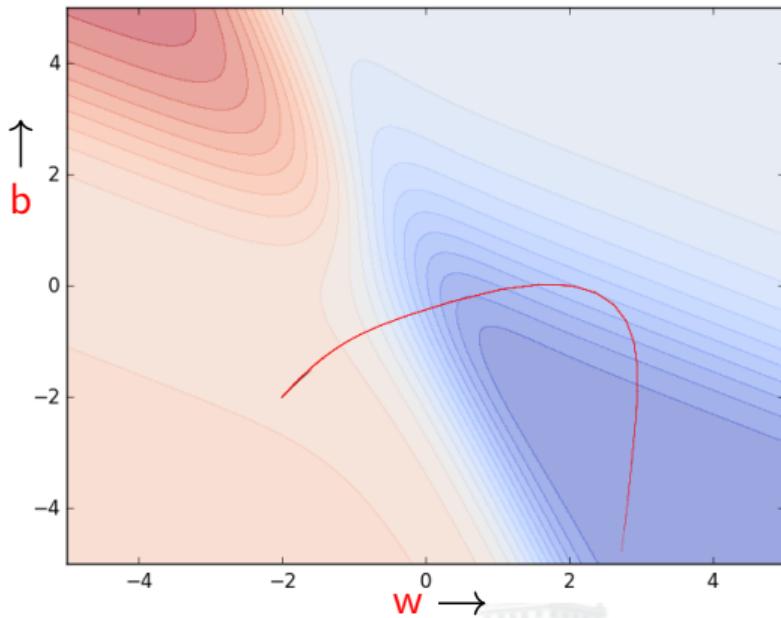
        v_w = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```



```

def do_momentum_gradient_descent() :
    w, b, eta = init_w, init_b, 1.0
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw
        v_b = gamma * prev_v_b + eta* db
        w = w - v_w
        b = b - v_b
        prev_v_w = v_w
        prev_v_b = v_b
    
```





思考

- 即使在平坦的区域，基于动量的梯度下降方法也能够快速更新



思考

- 即使在平坦的区域，基于动量的梯度下降方法也能够快速更新
- 快速更新就总是好吗？有没有可能导致越过目标点？



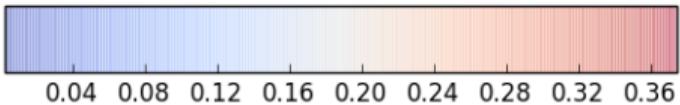
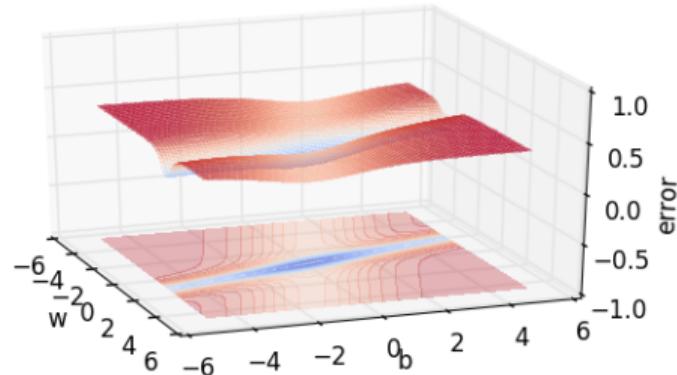
思考

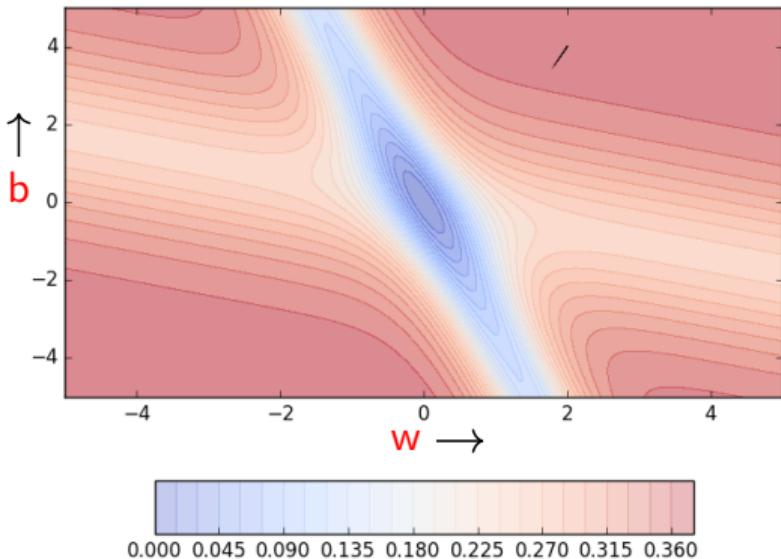
- 即使在平坦的区域，基于动量的梯度下降方法也能够快速更新
- 快速更新就总是好吗？有没有可能导致越过目标点？
- 看一个例子...

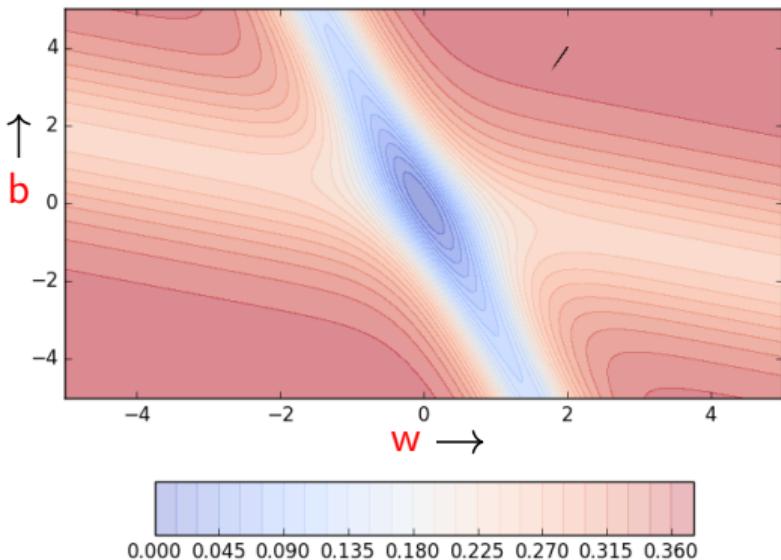


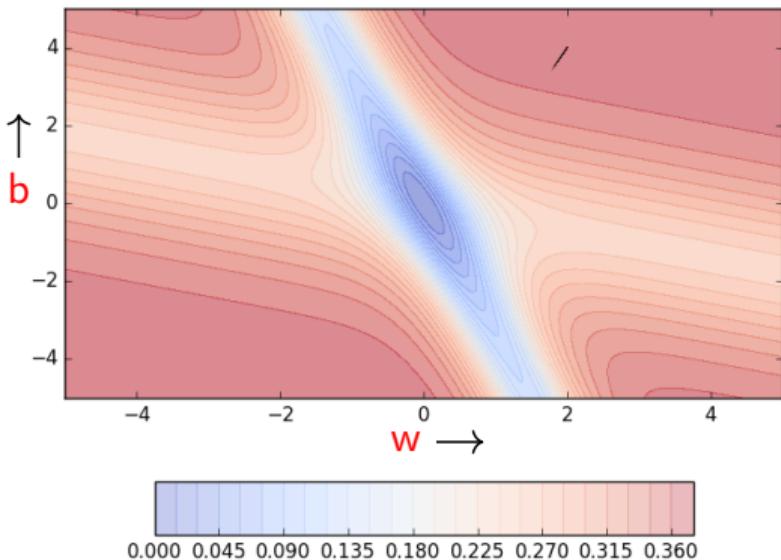
思考

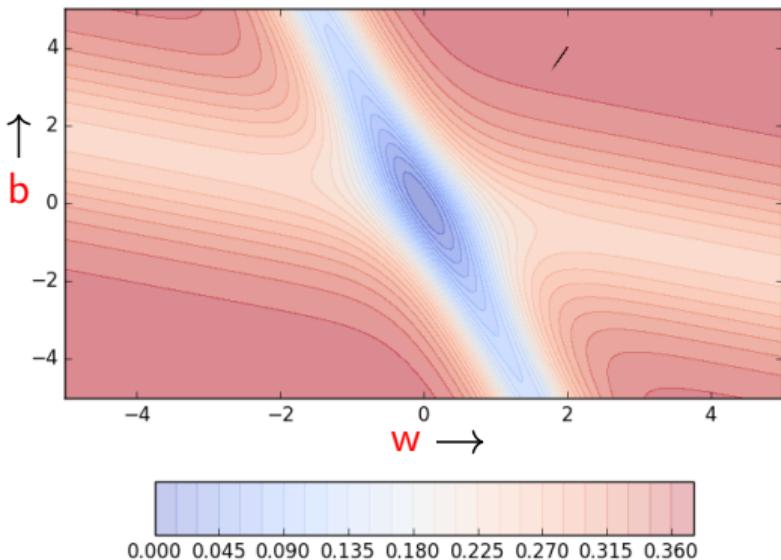
- 即使在平坦的区域，基于动量的梯度下降方法也能够快速更新
- 快速更新就总是好吗？有没有可能导致越过目标点？
- 看一个例子...

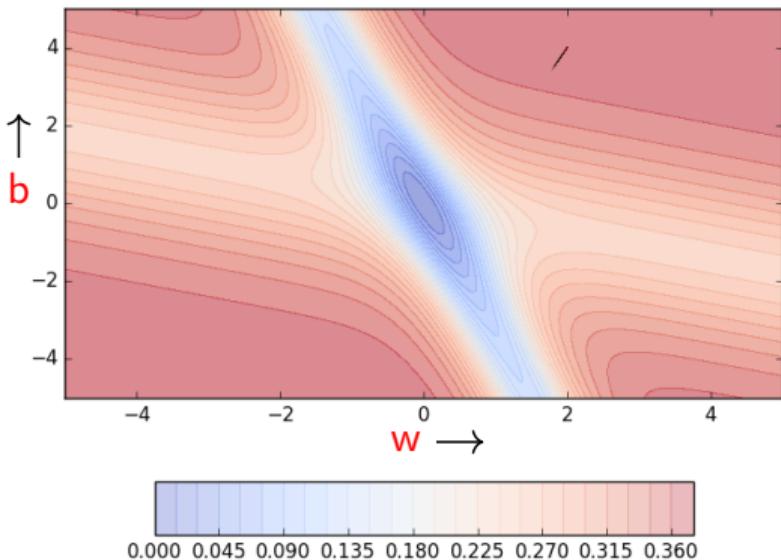


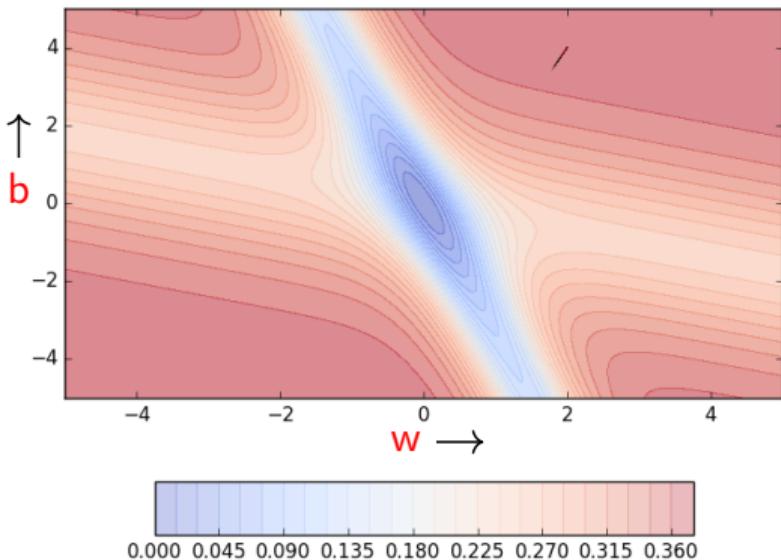


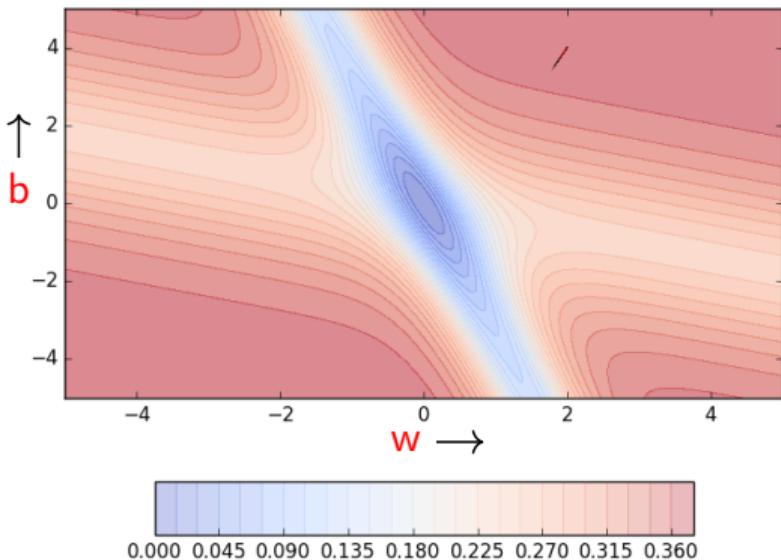


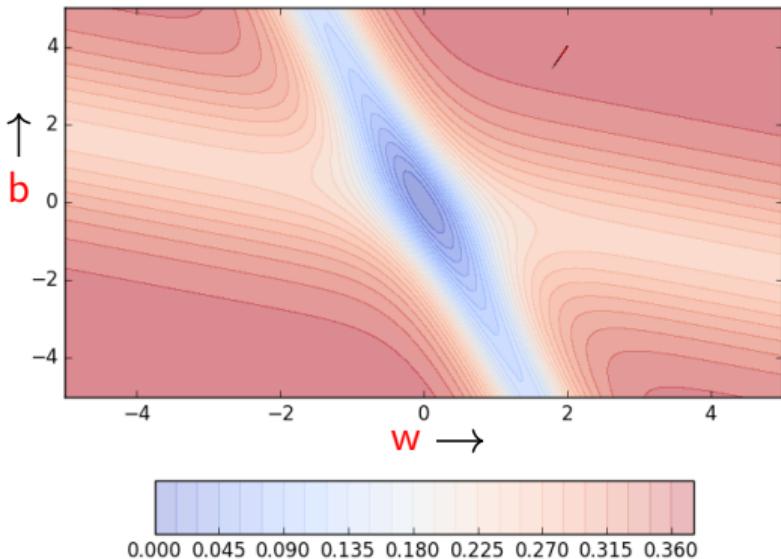


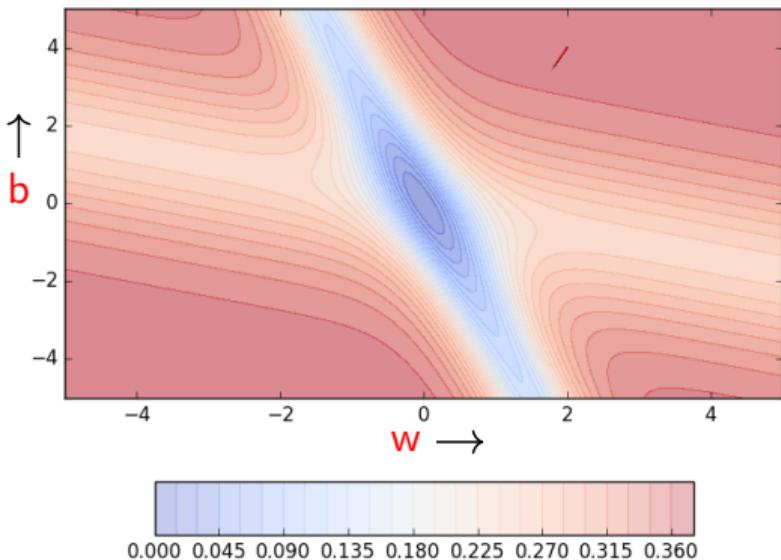


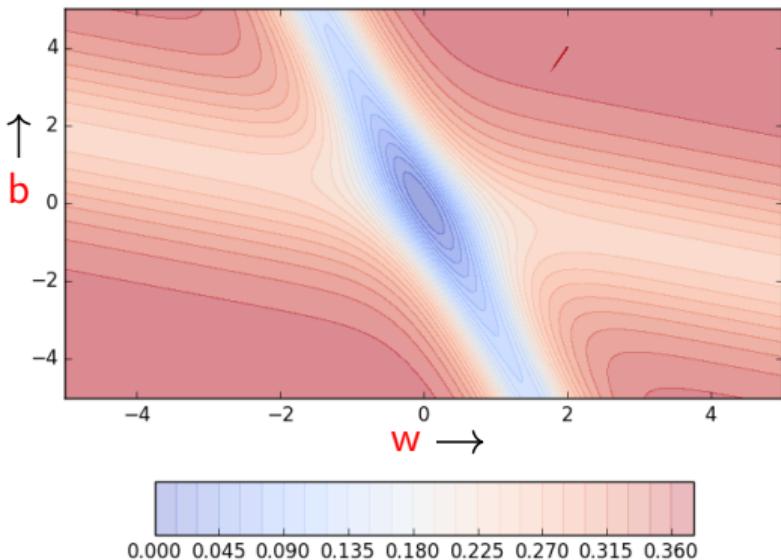


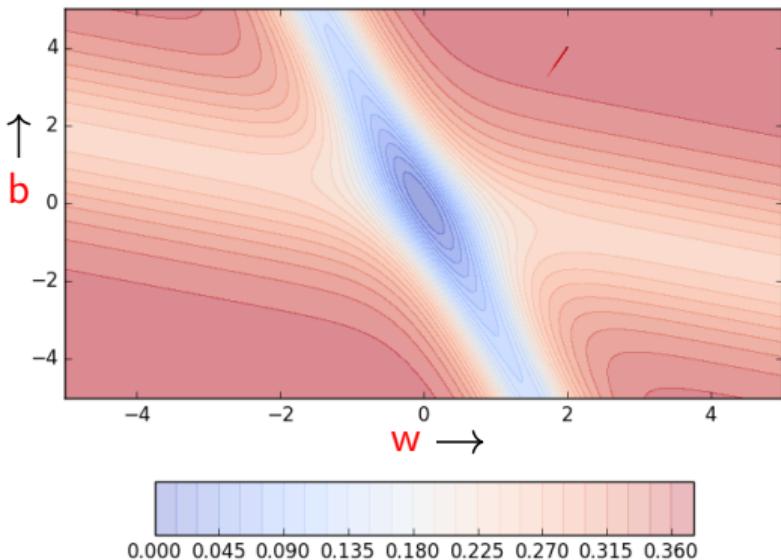


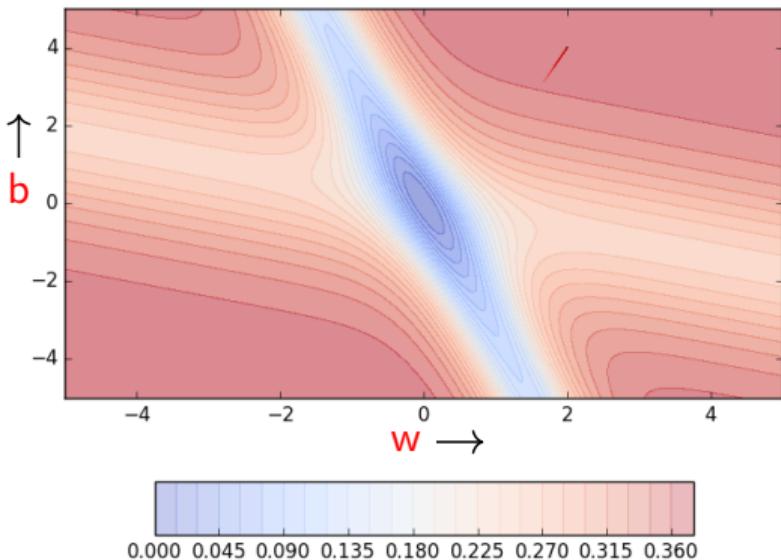


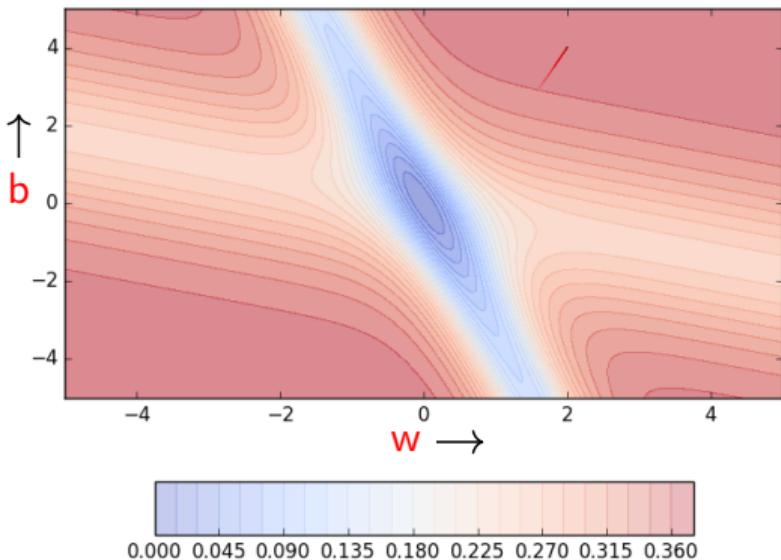


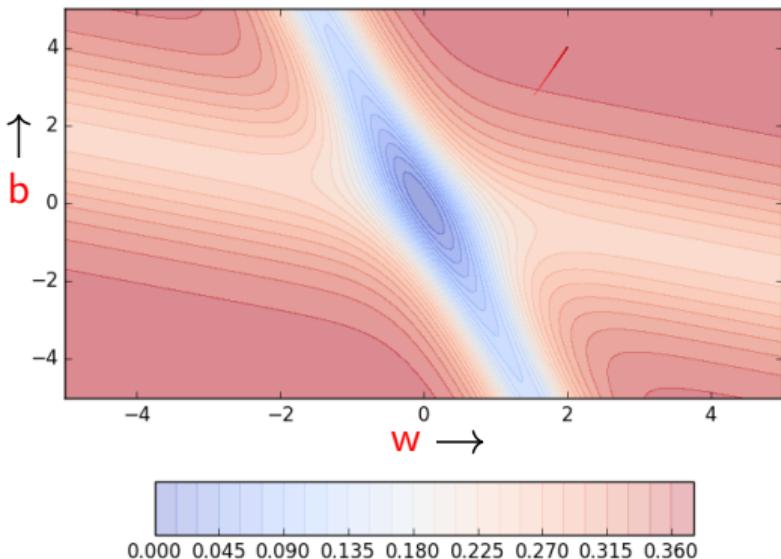


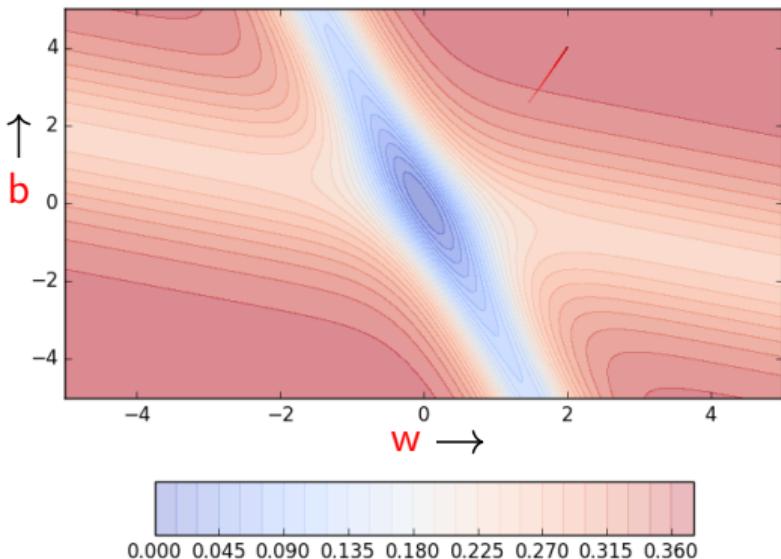


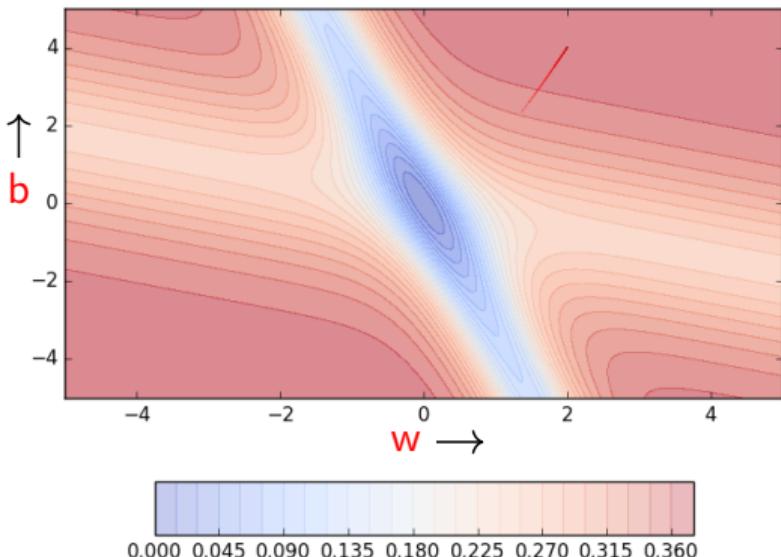


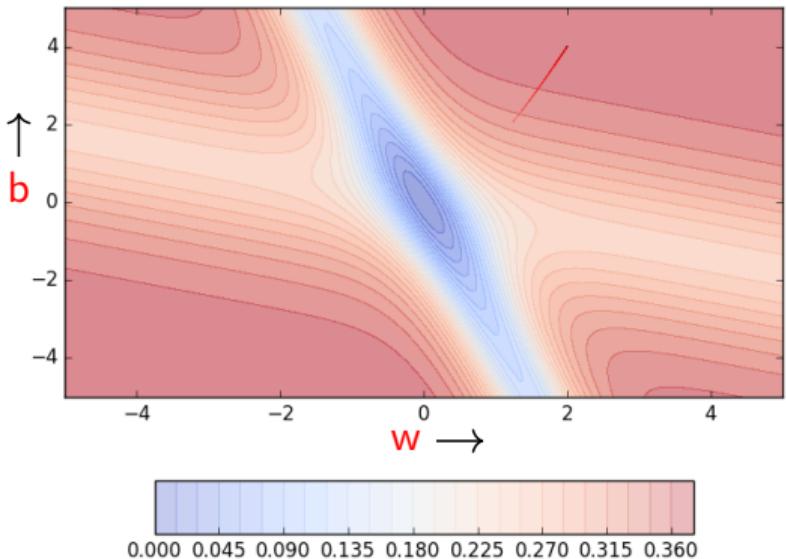


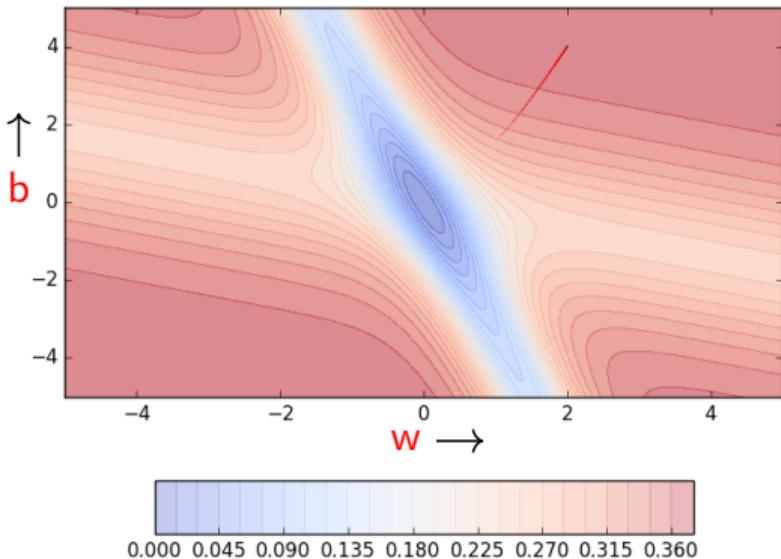


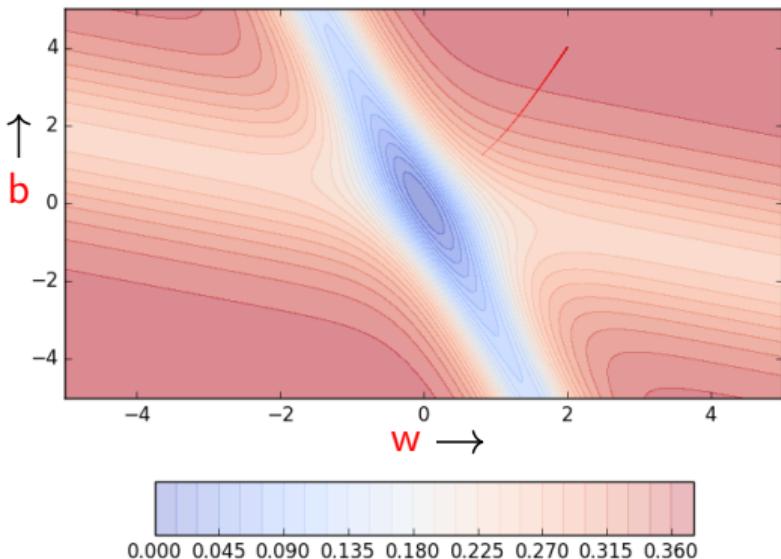


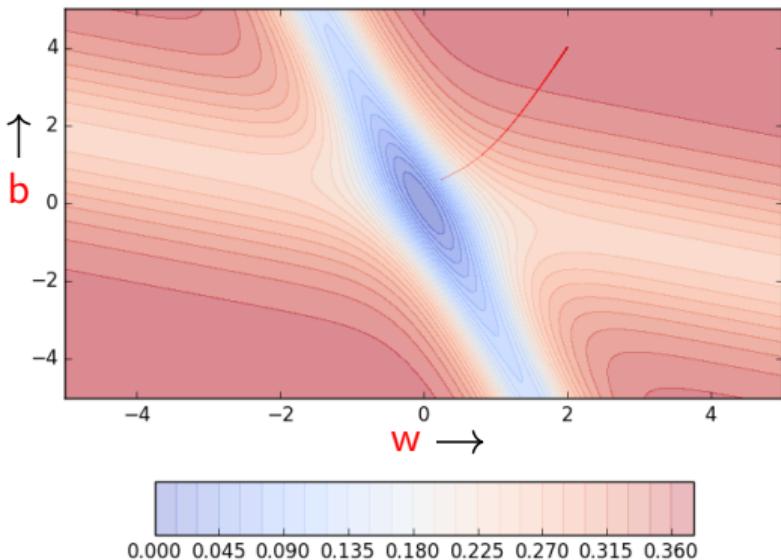


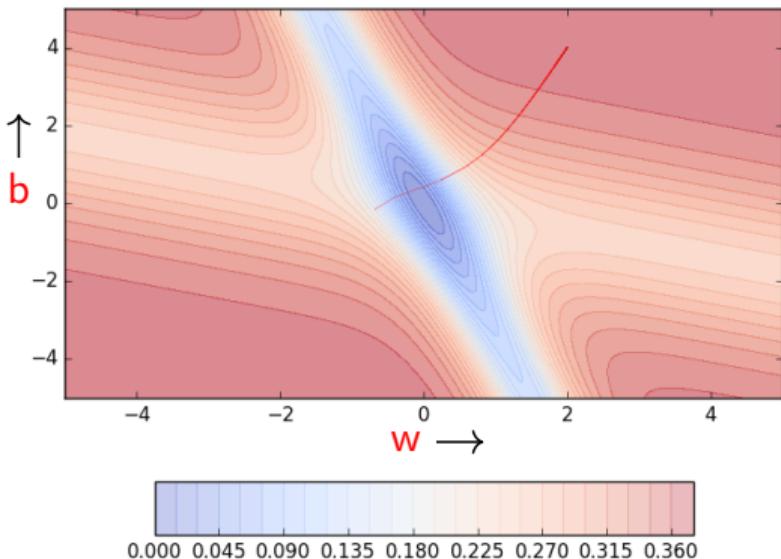


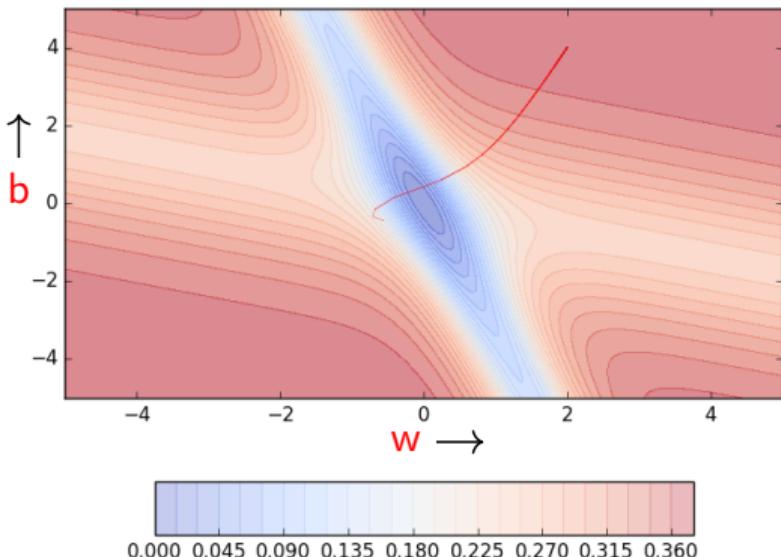


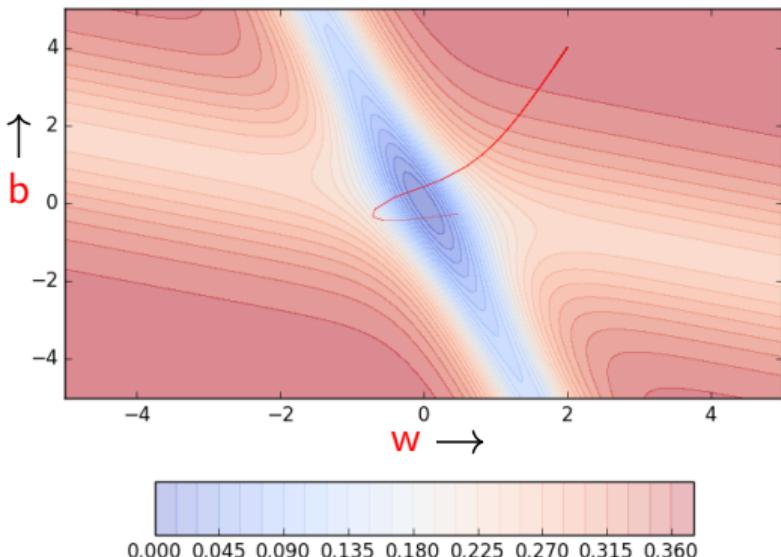


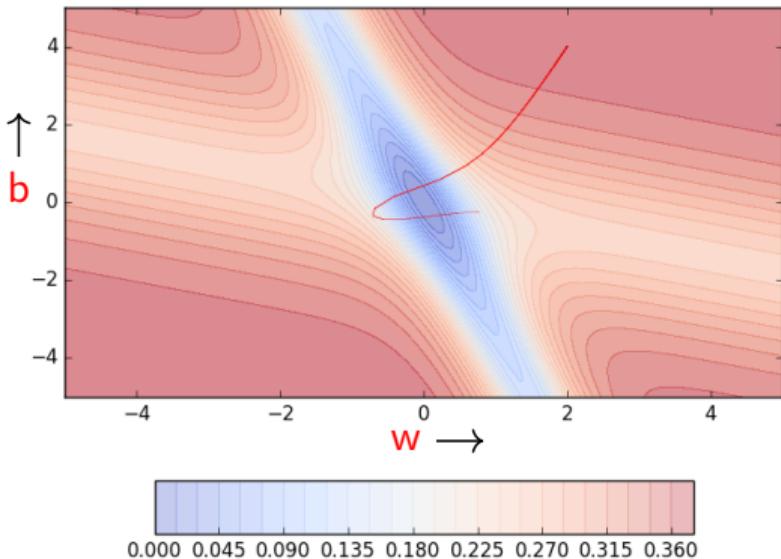


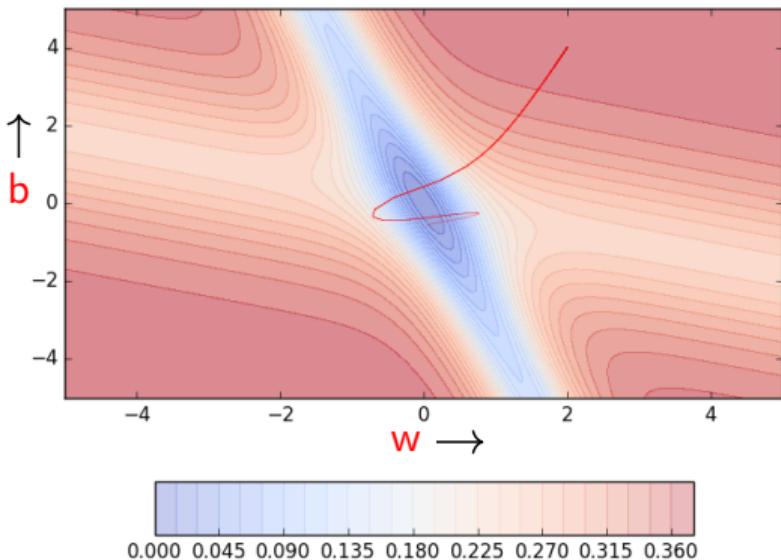


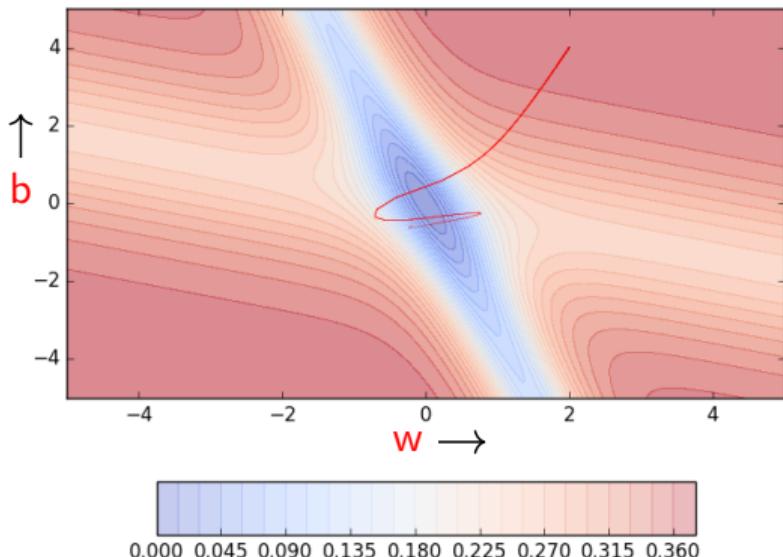


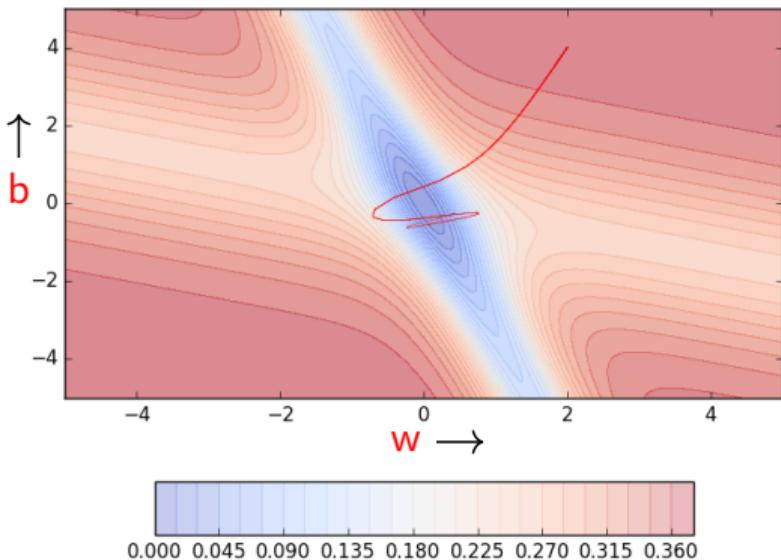


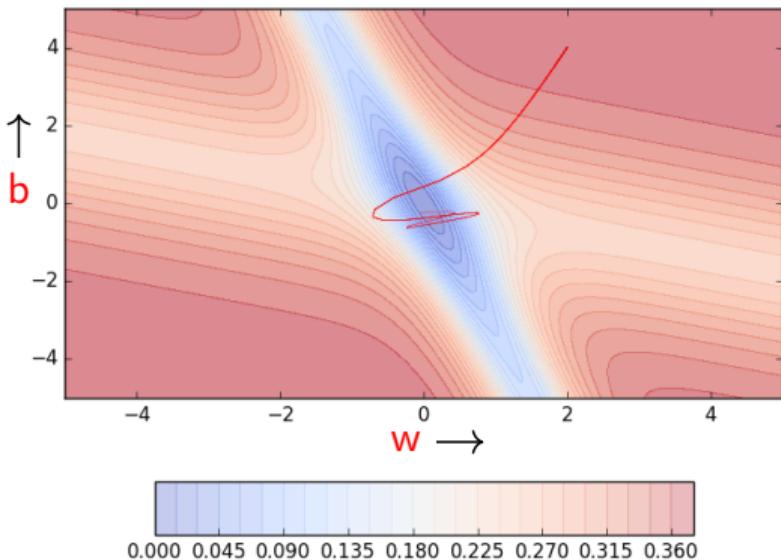


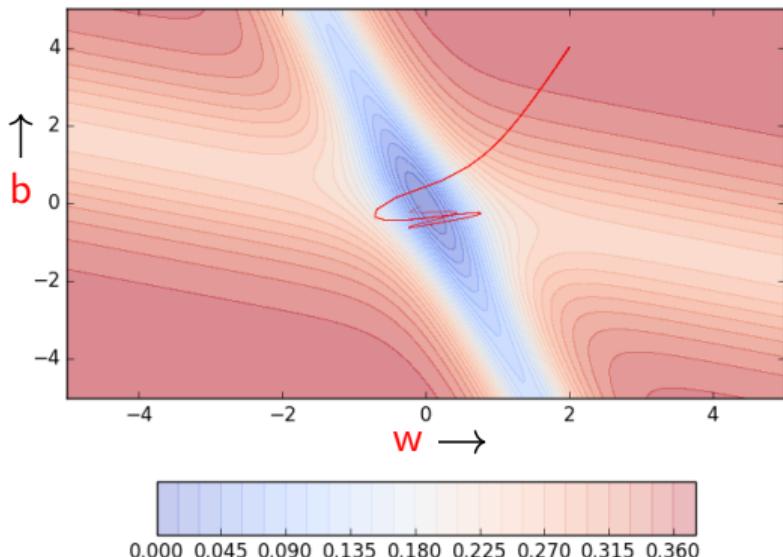


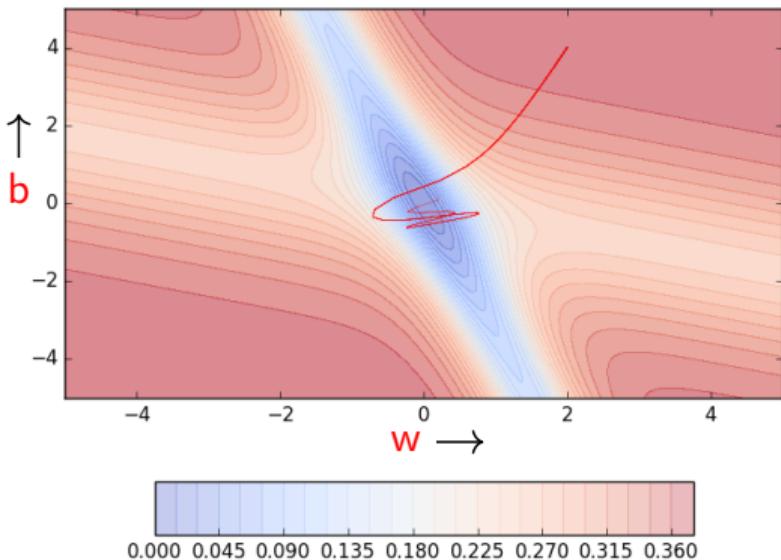


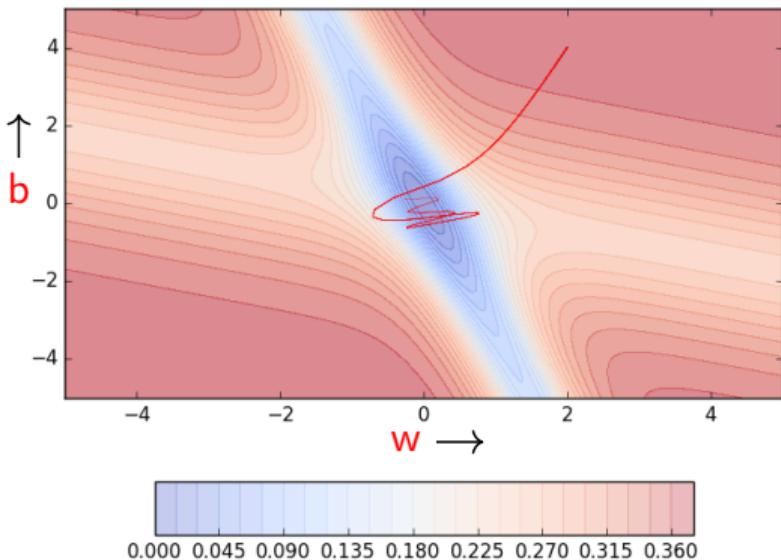


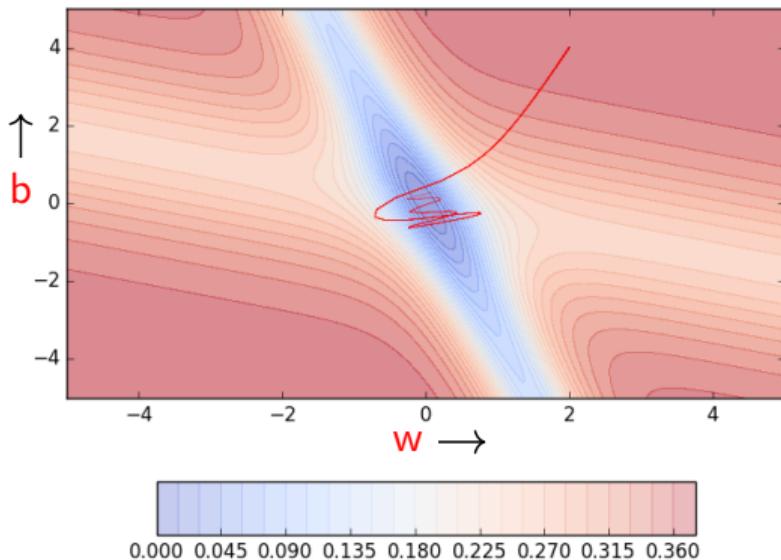


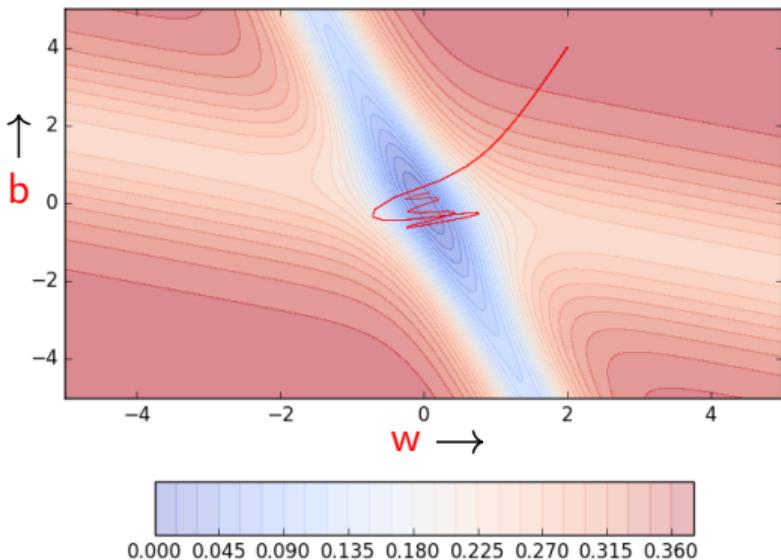


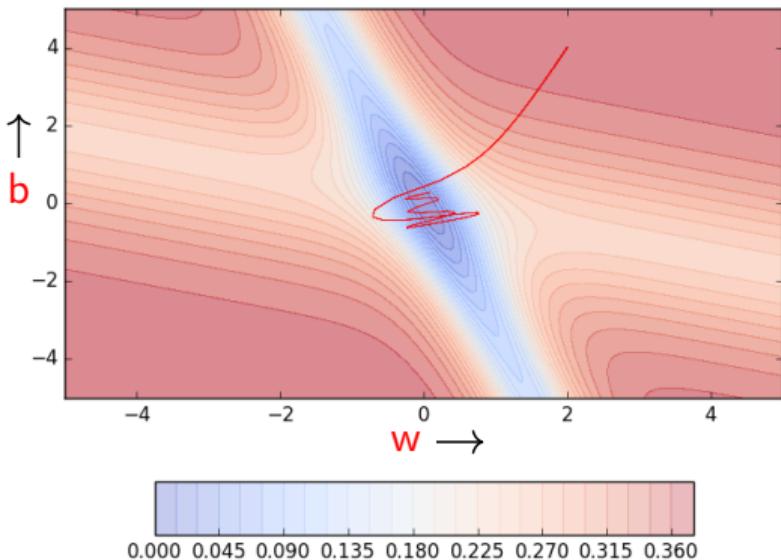


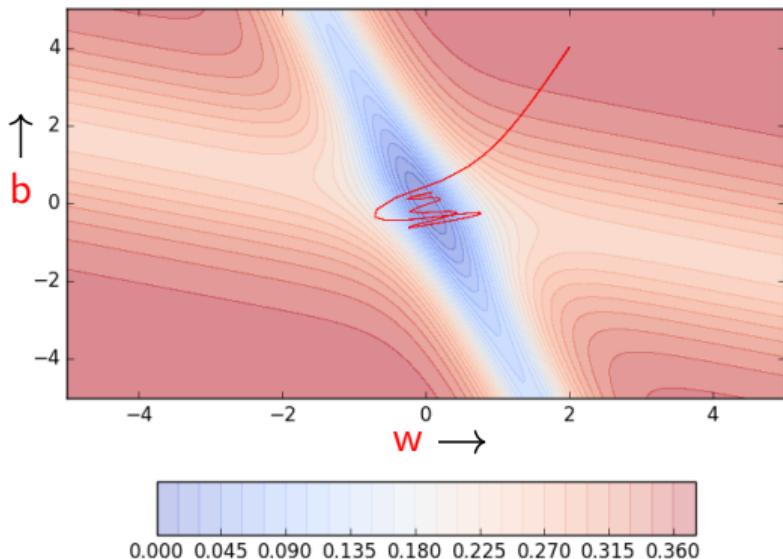


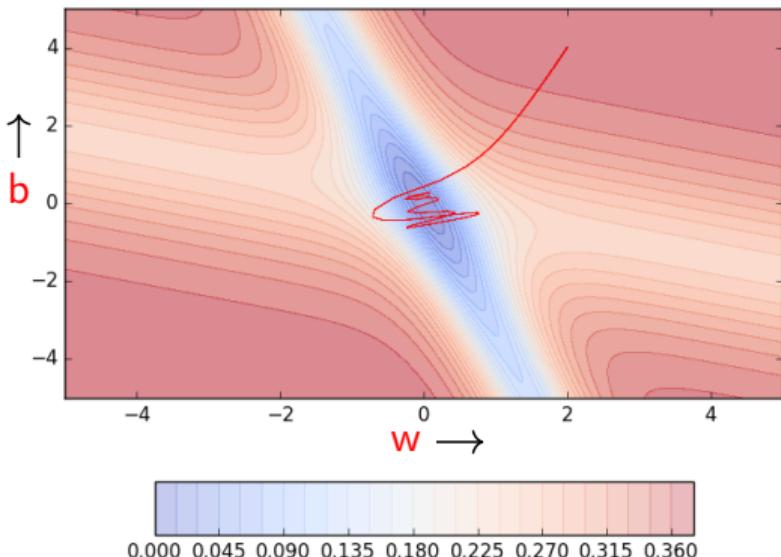


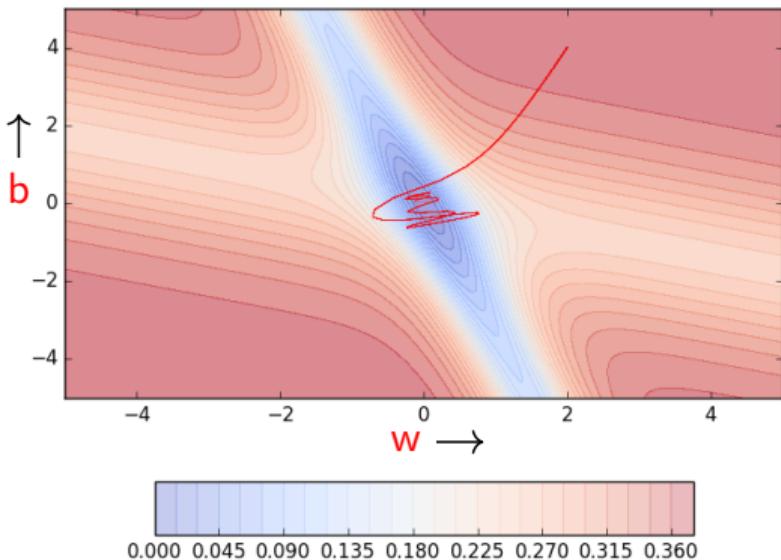




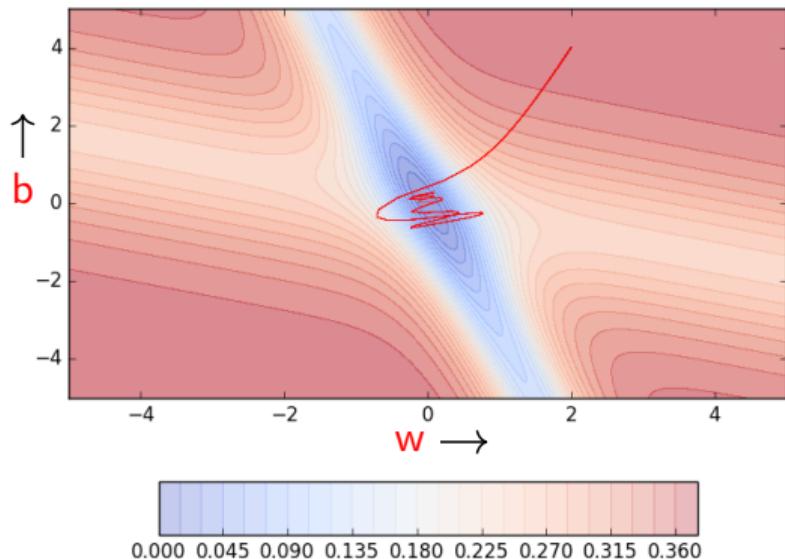




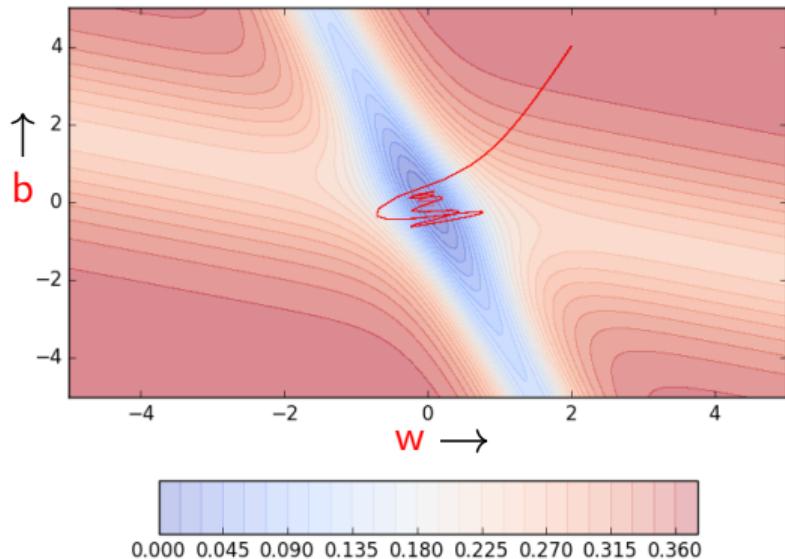




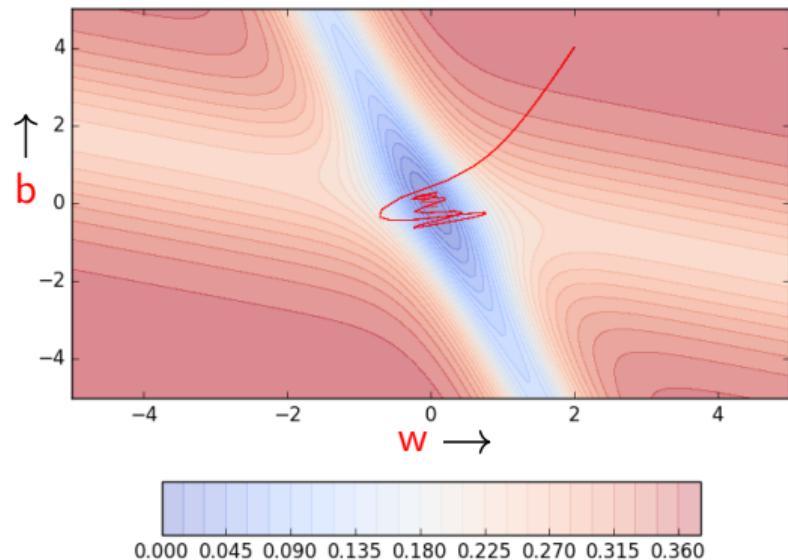
- 基于动量的梯度下降方法在峡谷处来回振荡



- 基于动量的梯度下降方法在峡谷处来回振荡
- 尽量多次振荡，仍然比标准梯度下降算法收敛的更快

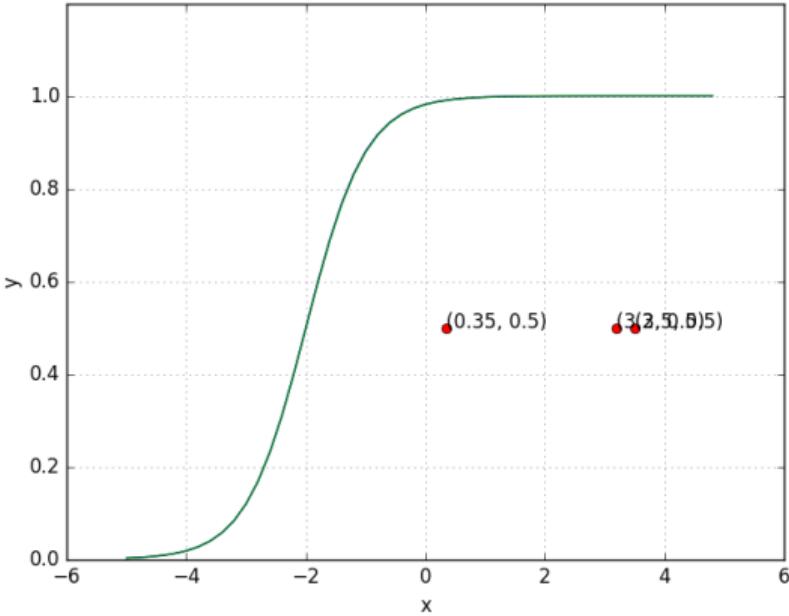
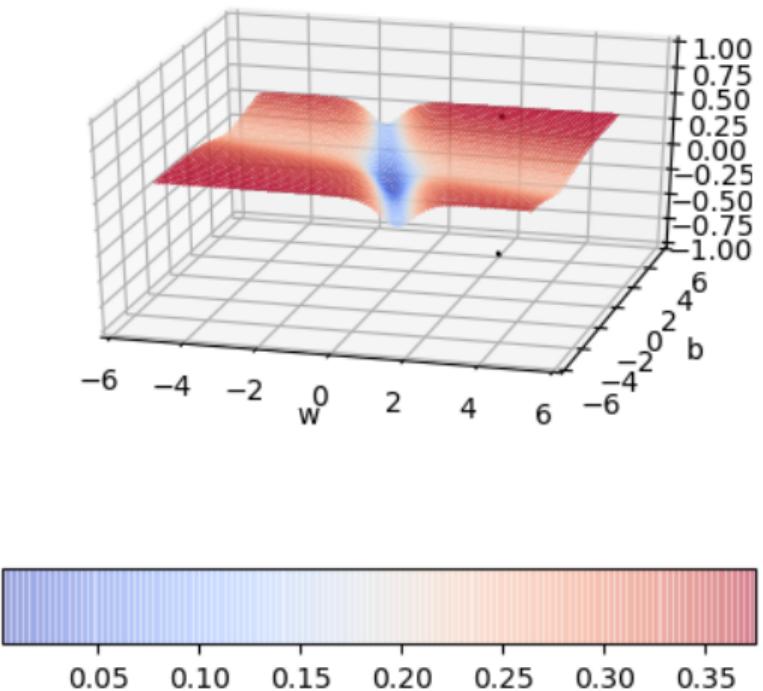


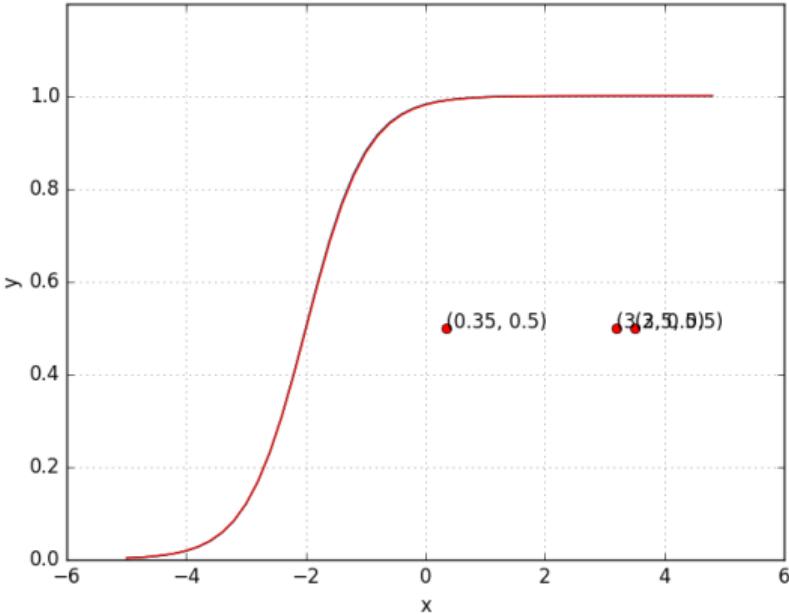
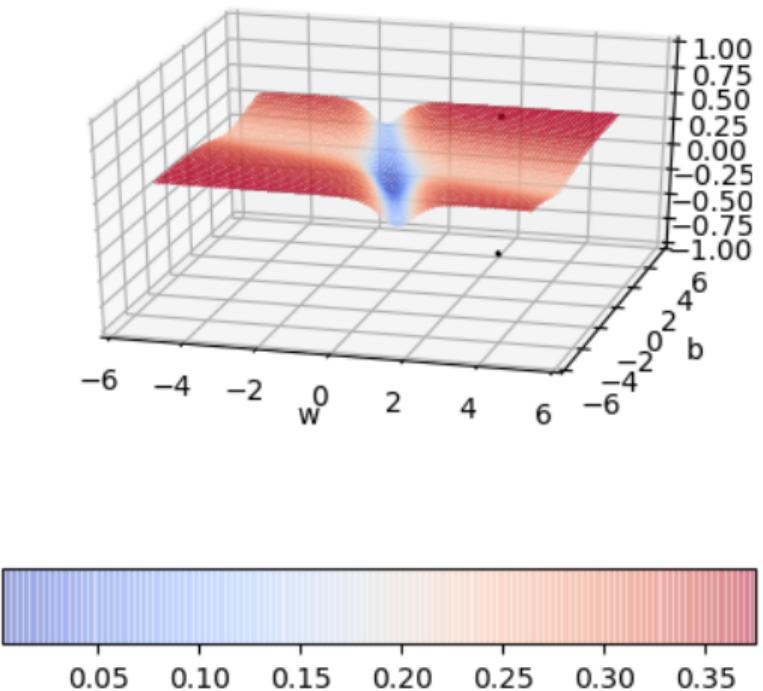
- 基于动量的梯度下降方法在峡谷处来回振荡
- 尽量多次振荡，仍然比标准梯度下降算法收敛的更快
- 100 次迭代后，基于动量的梯度下降方法将损失降到 0.00001，但标准梯度下降方法困在了 0.36 处

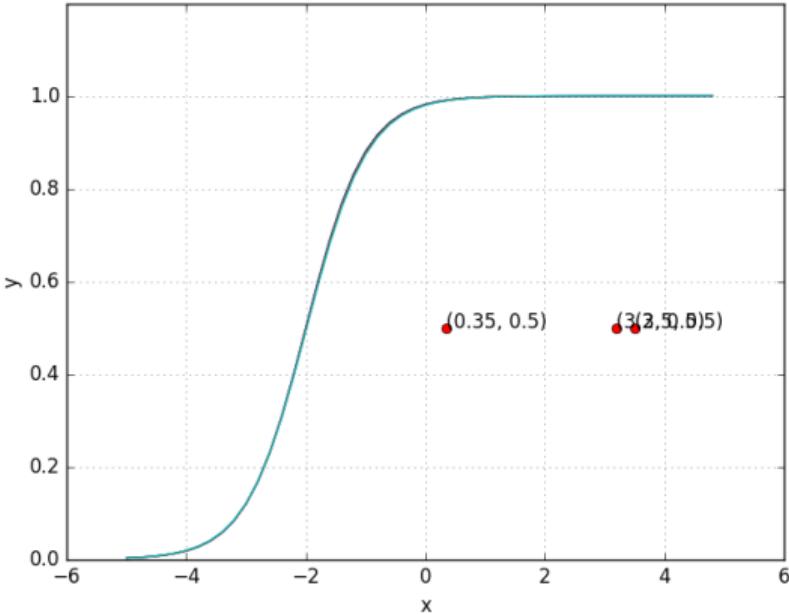
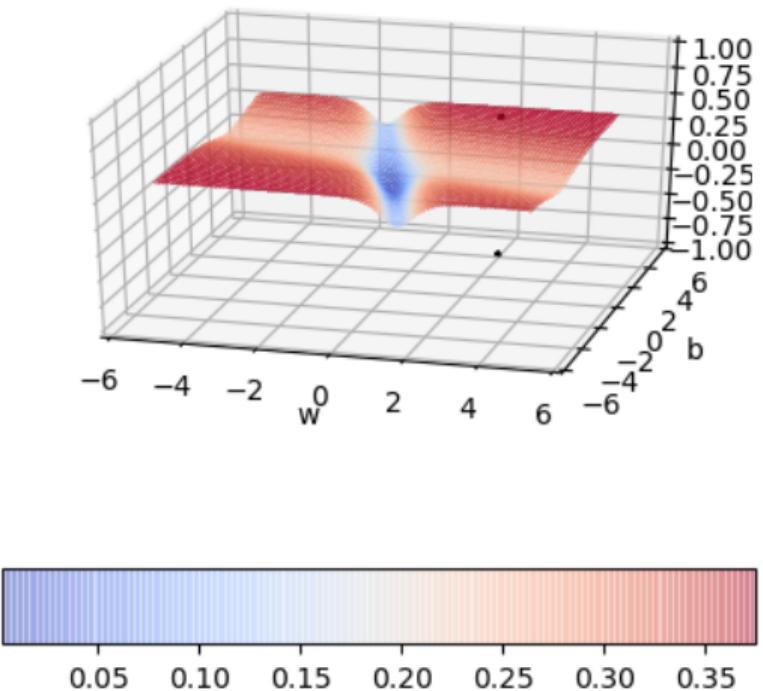


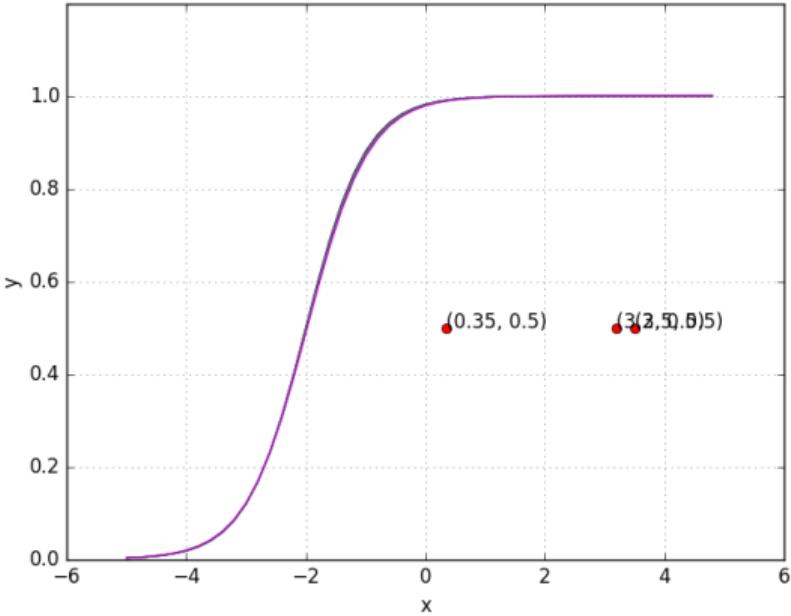
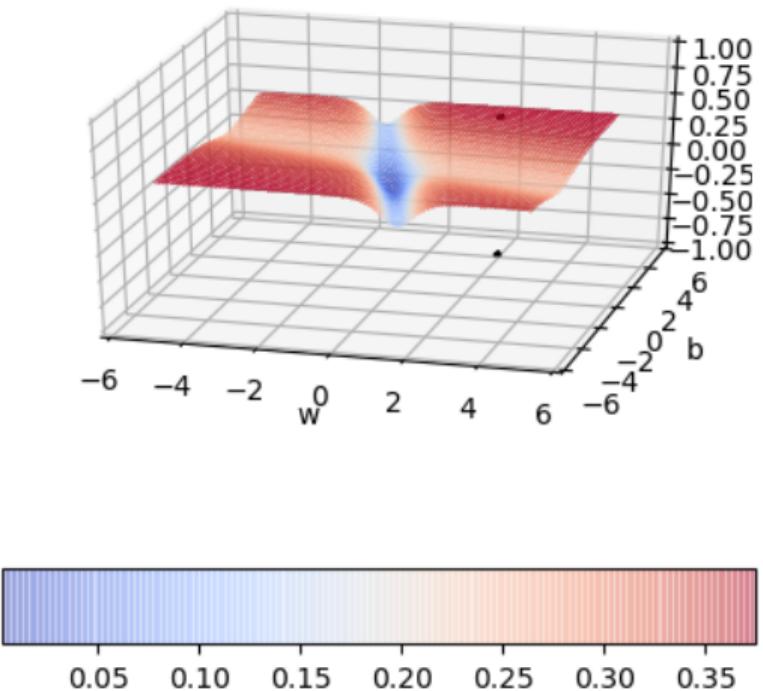


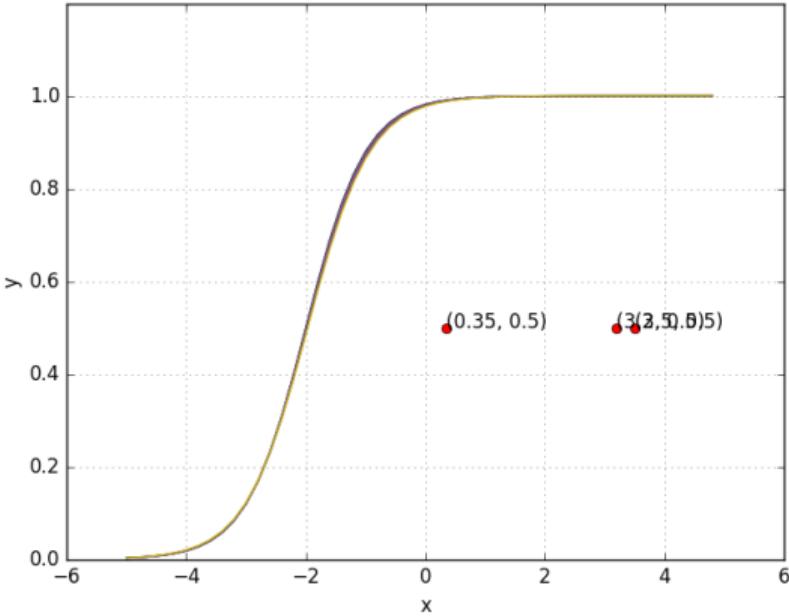
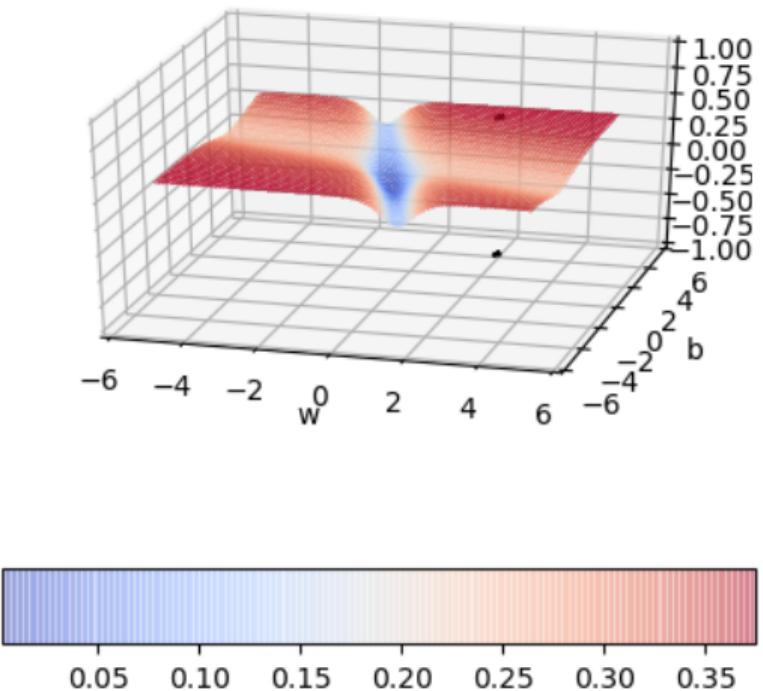
借助 3D 可视化观察上述过程的几何解释

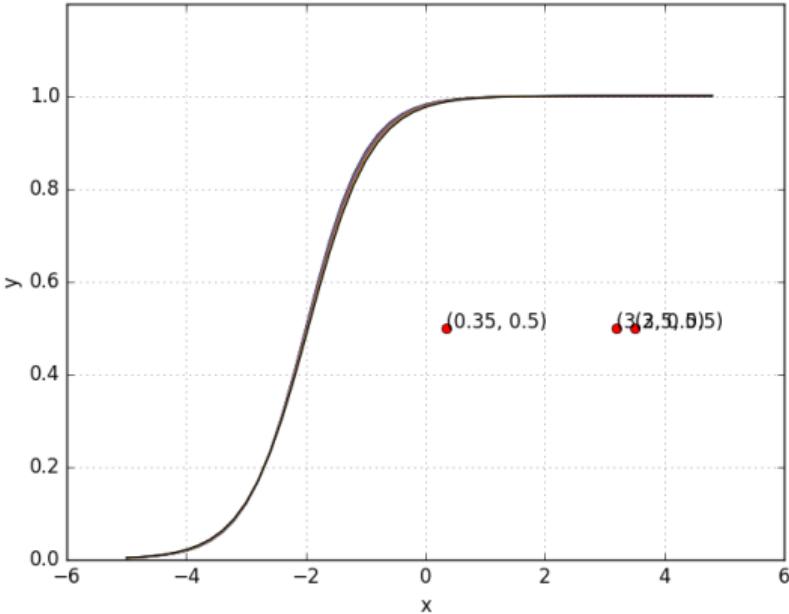
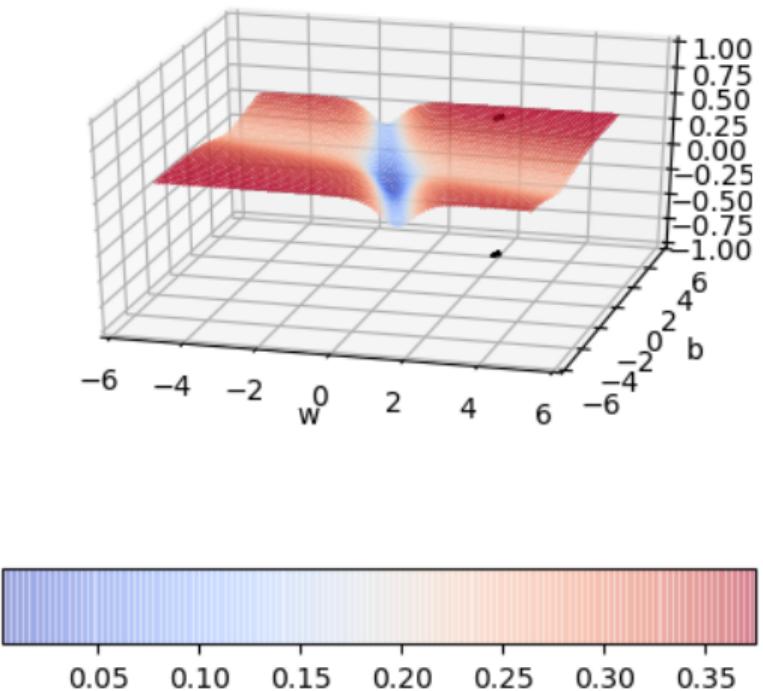


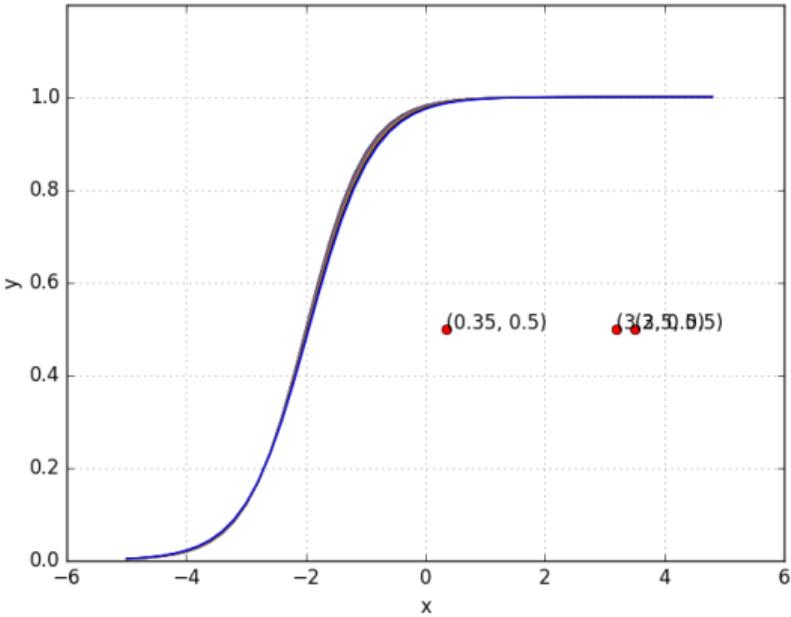
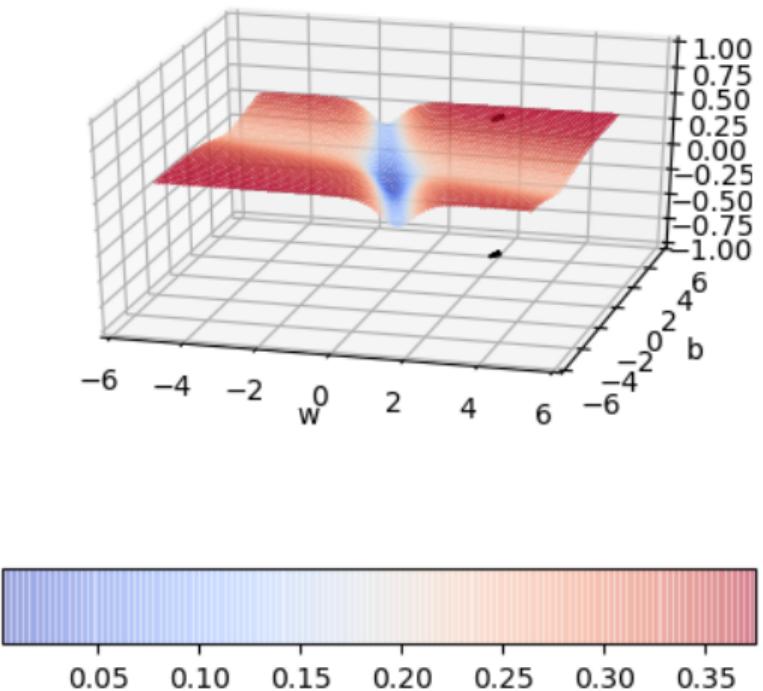


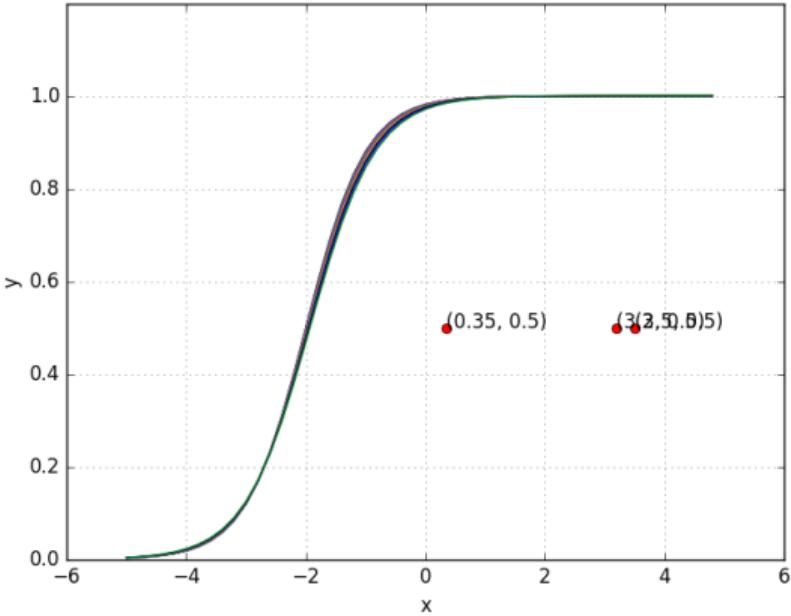
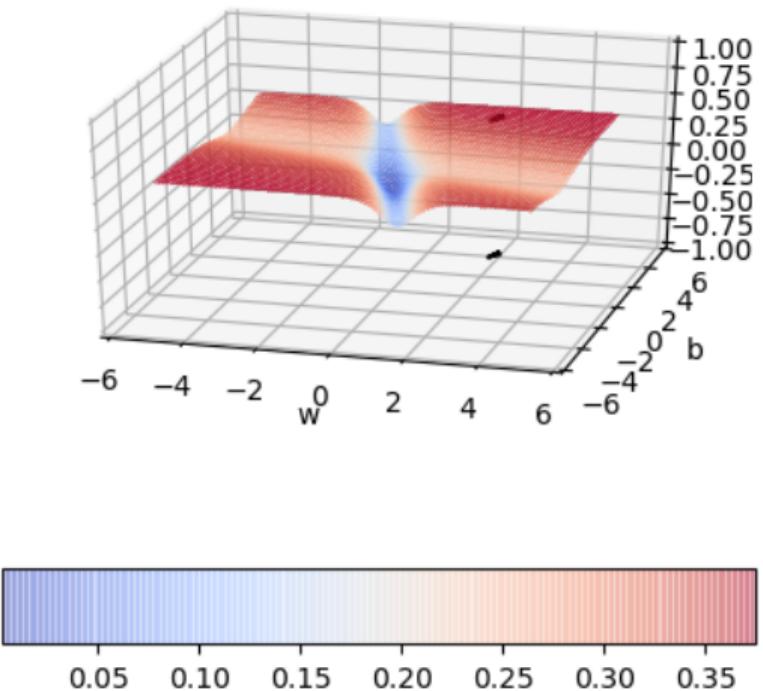


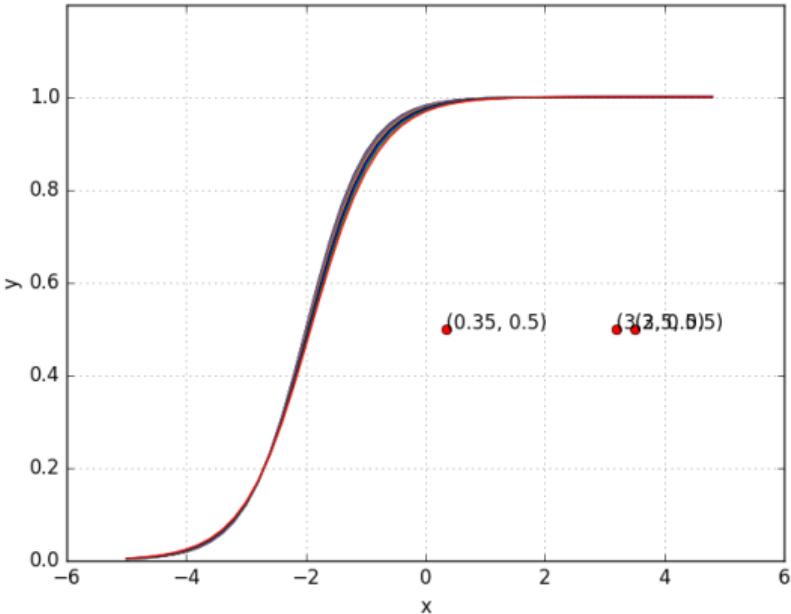
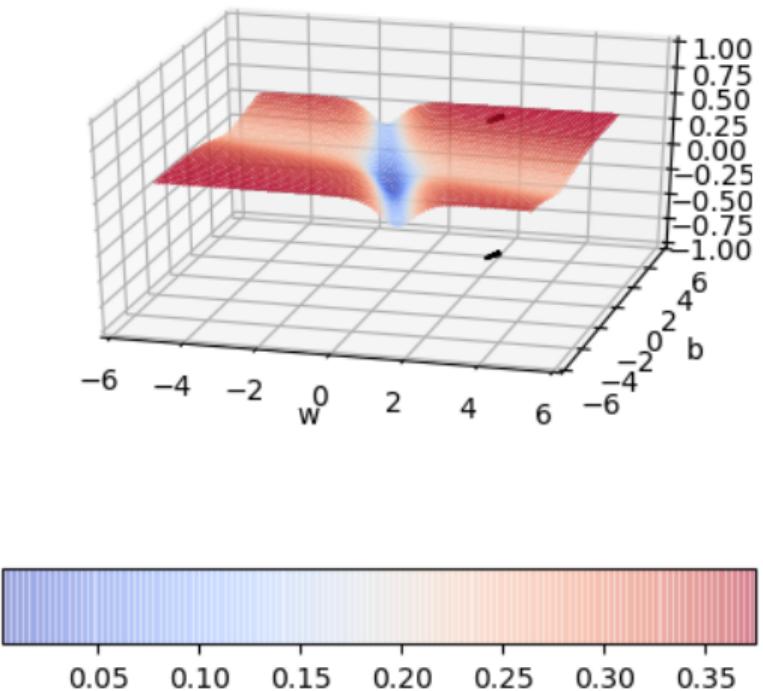


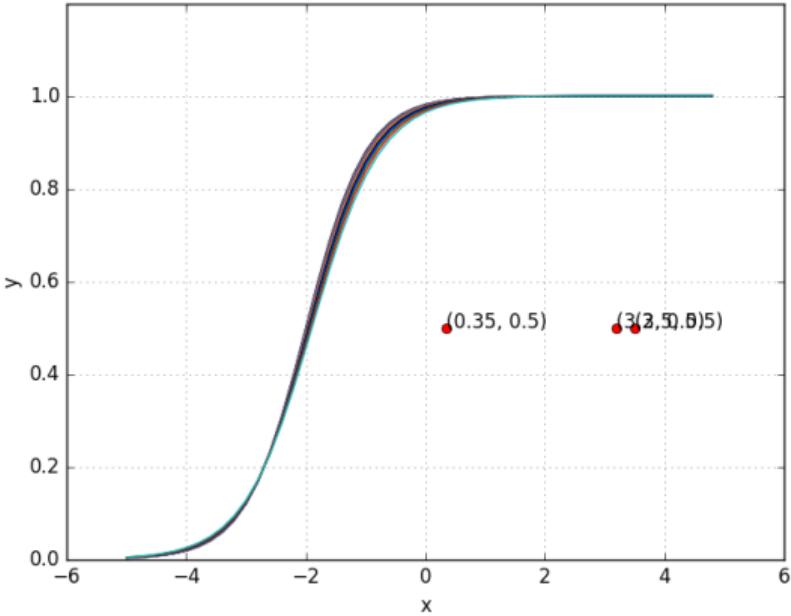
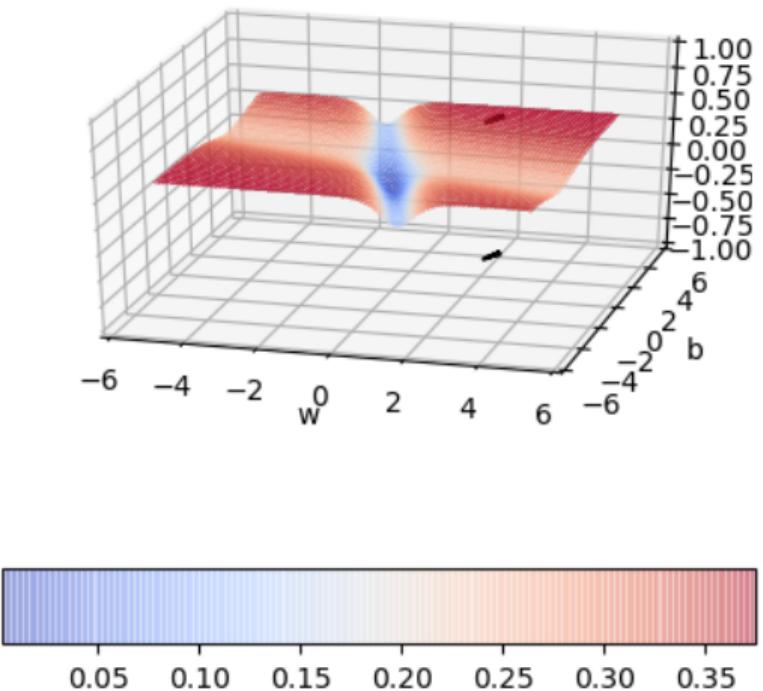


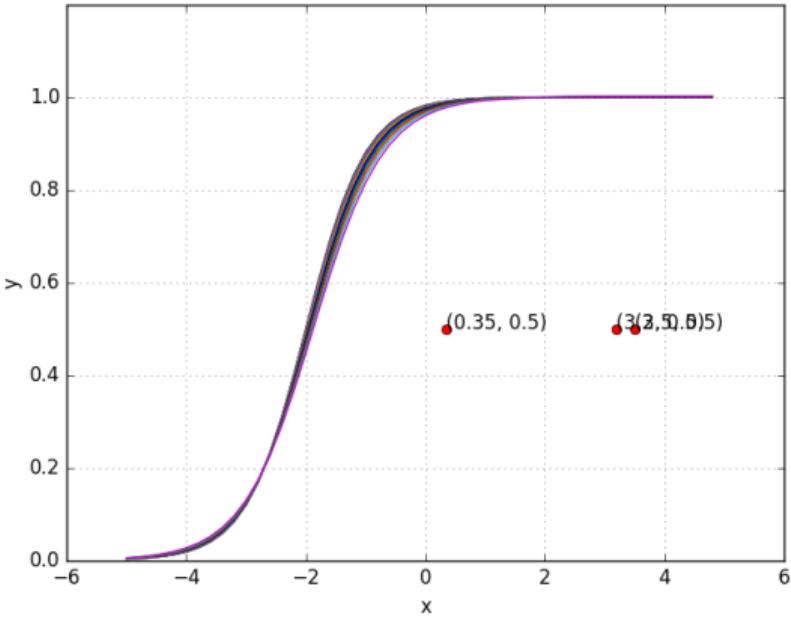
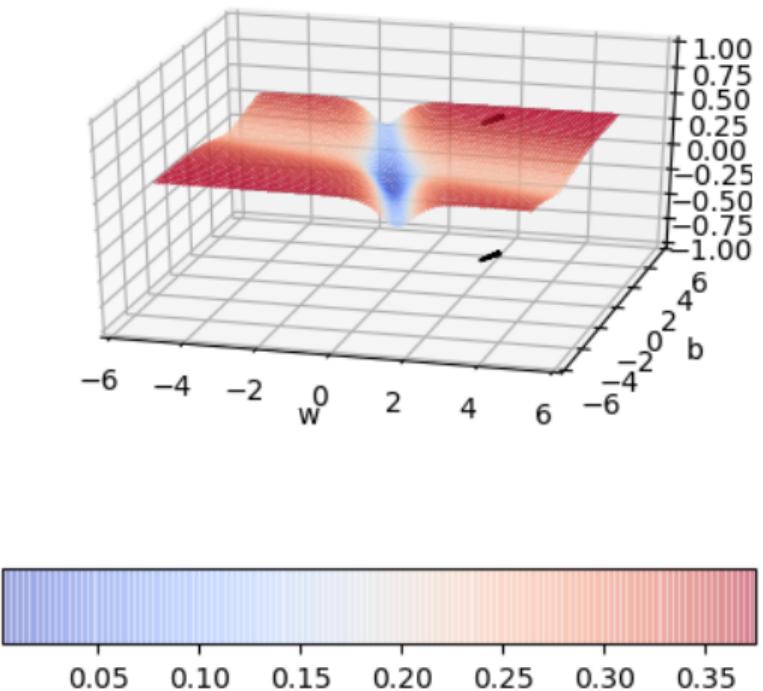


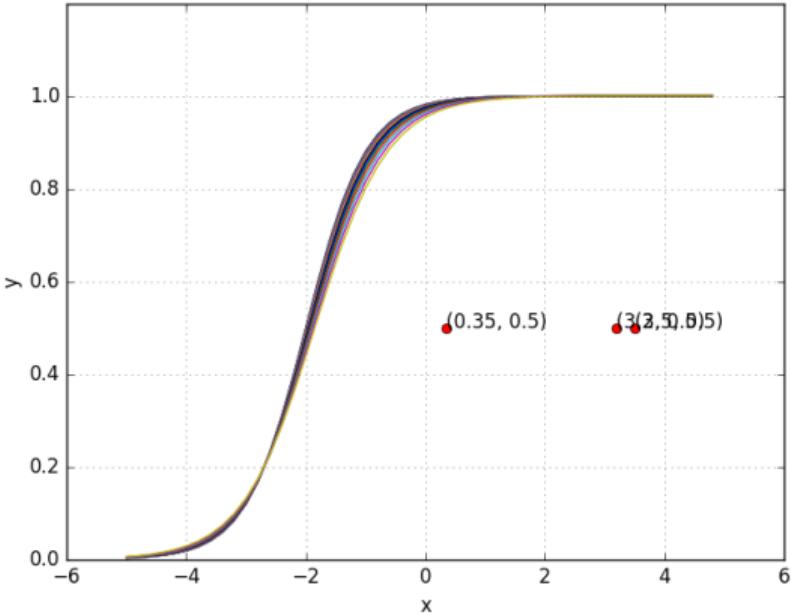
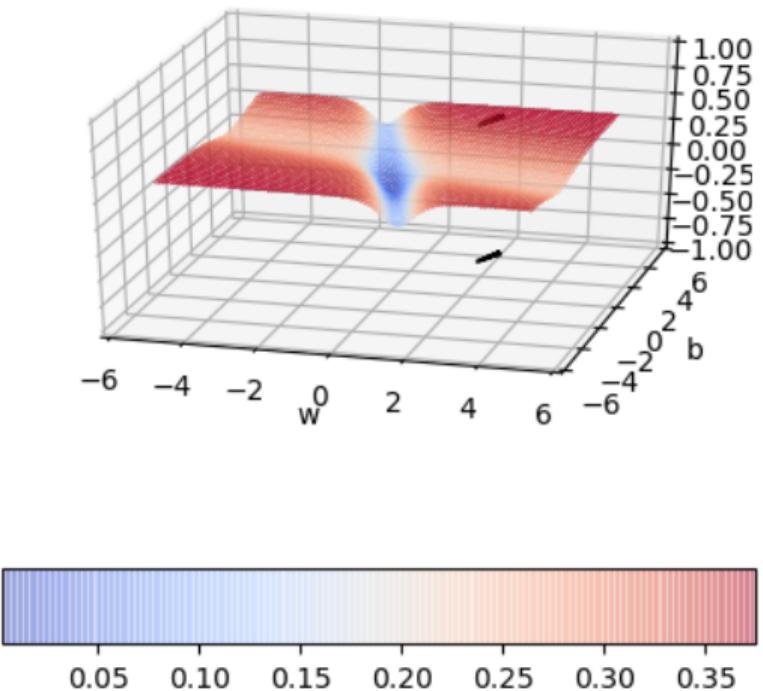


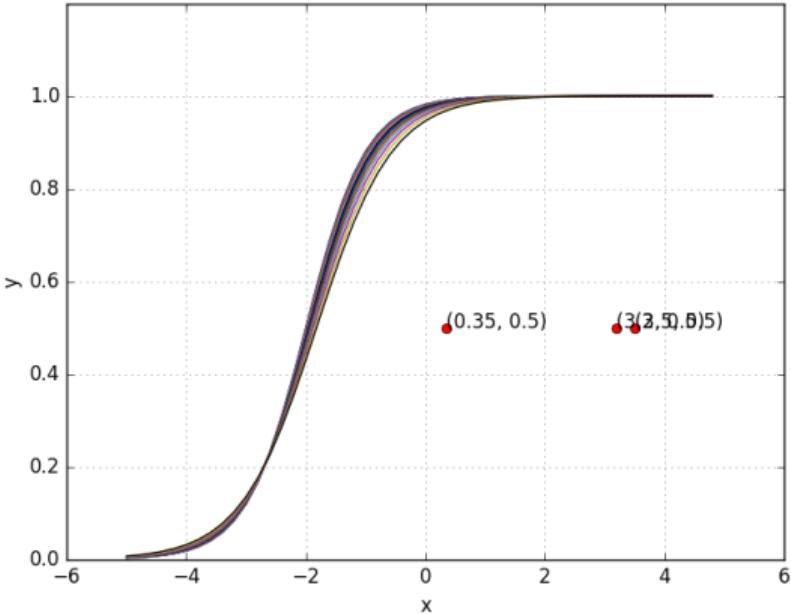
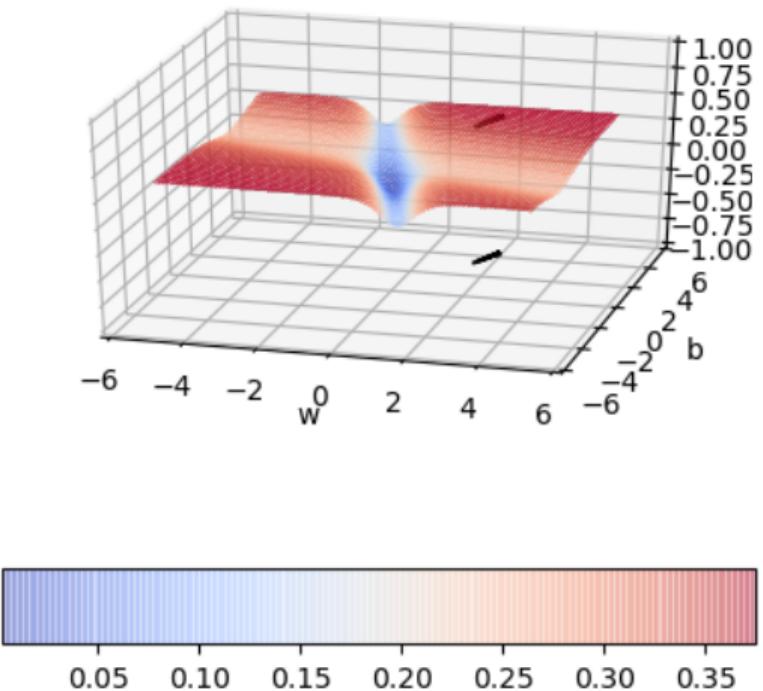


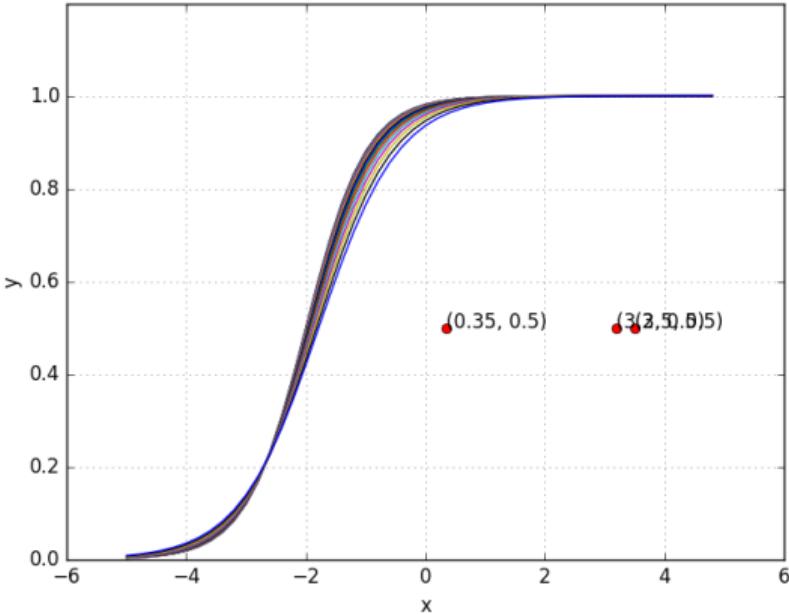
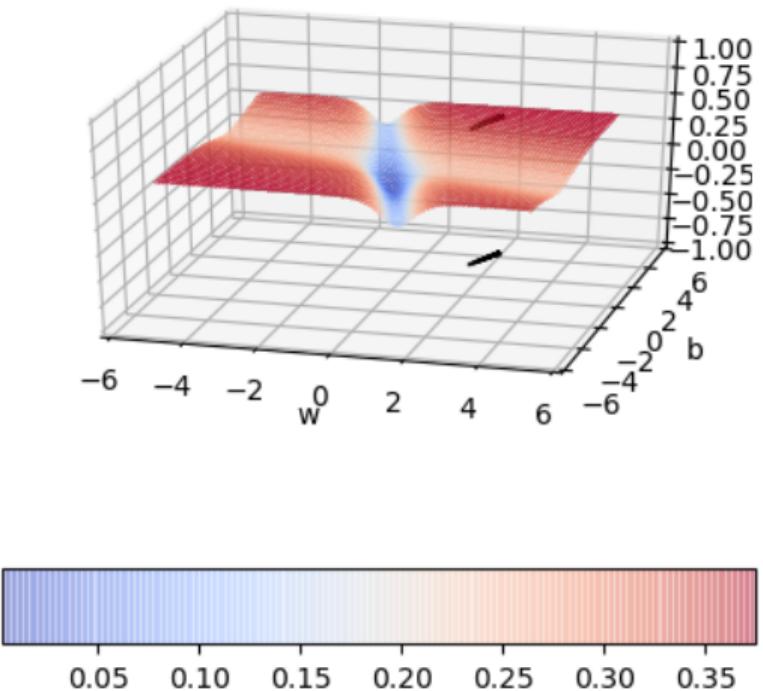


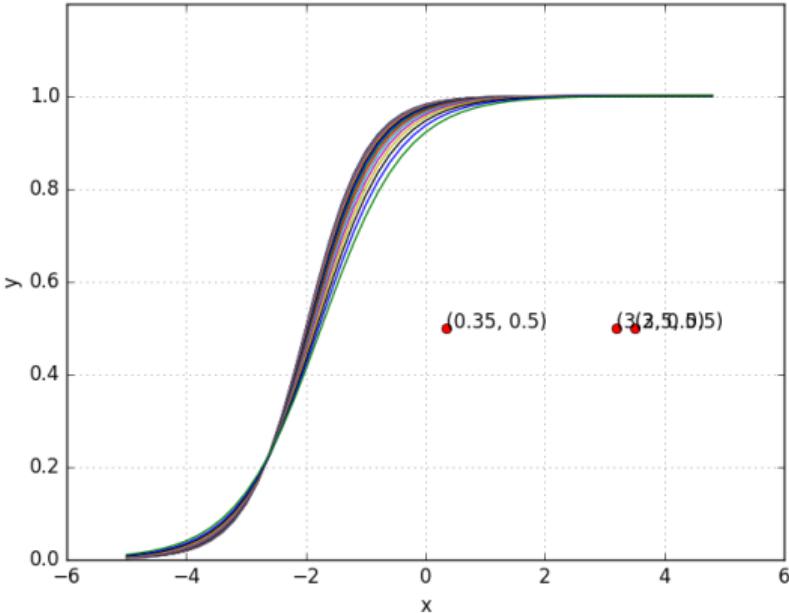
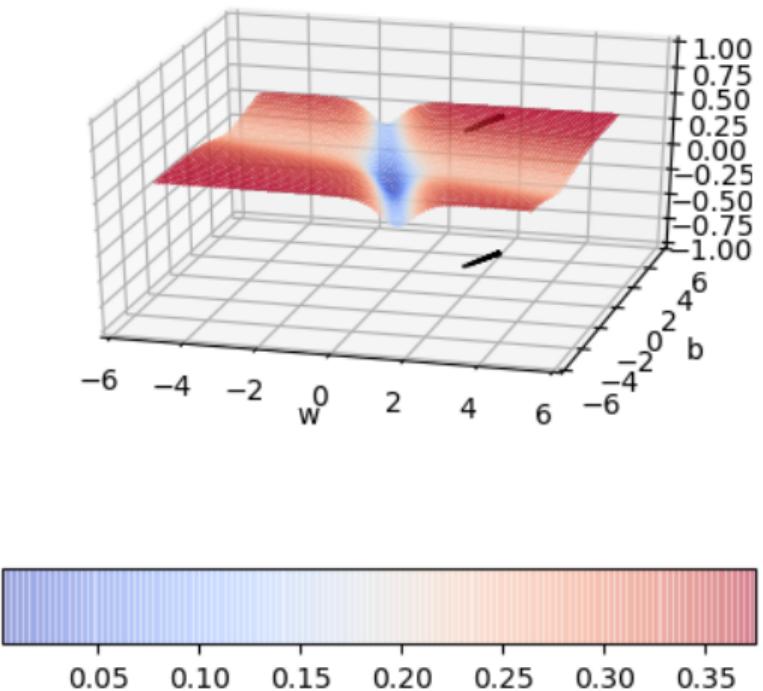


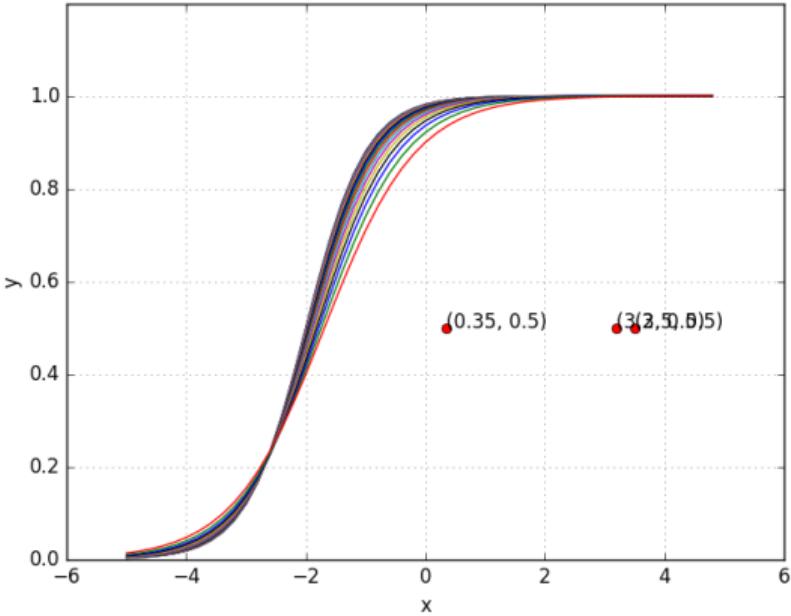
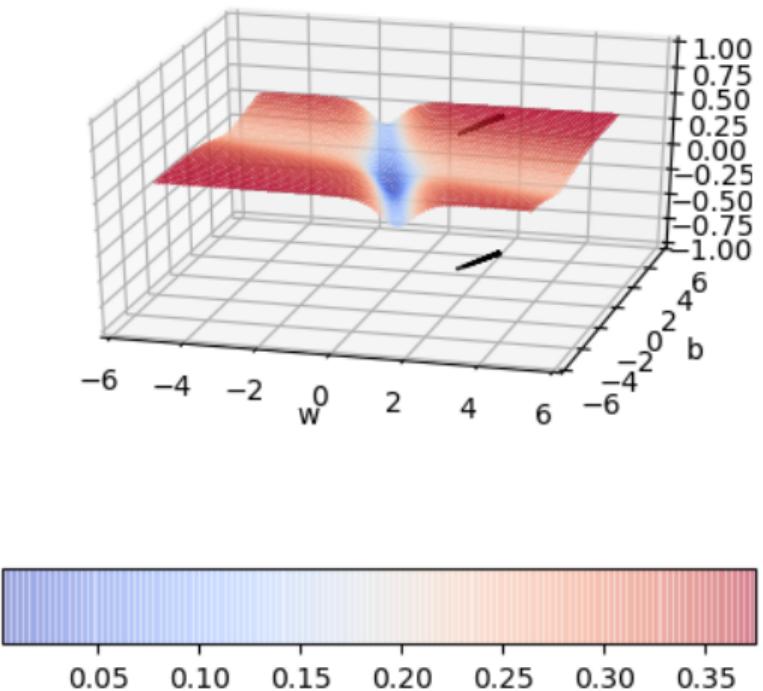


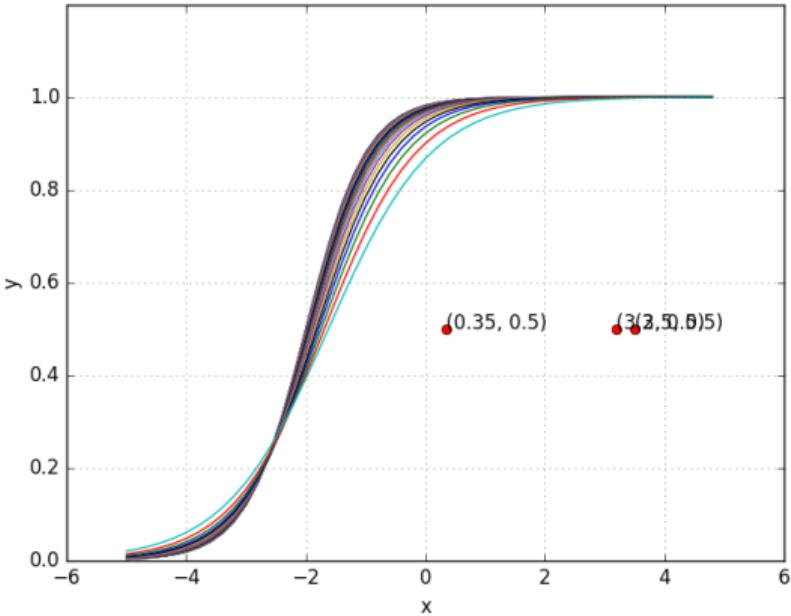
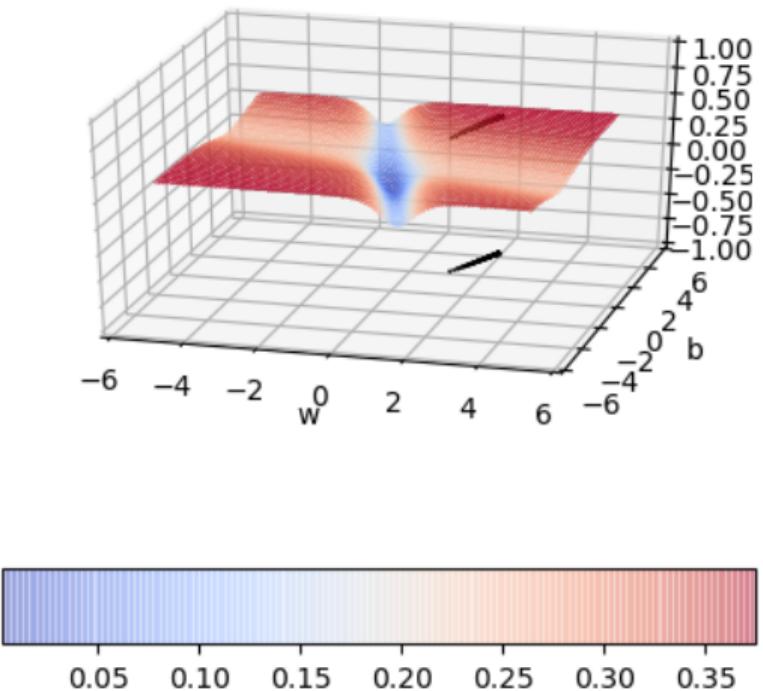


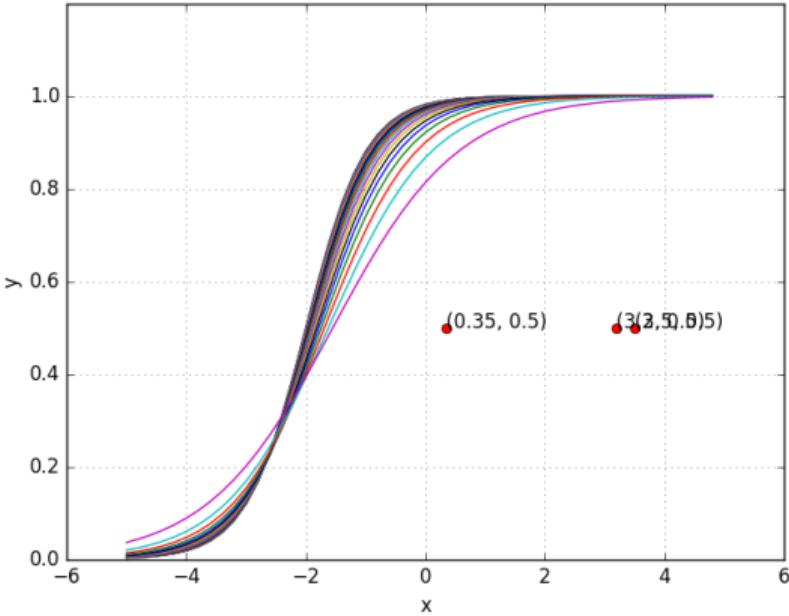
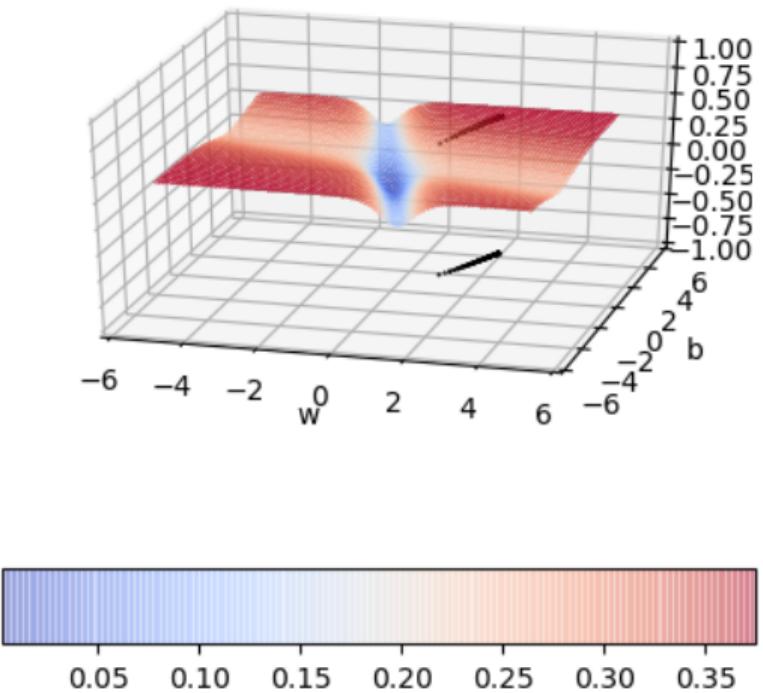


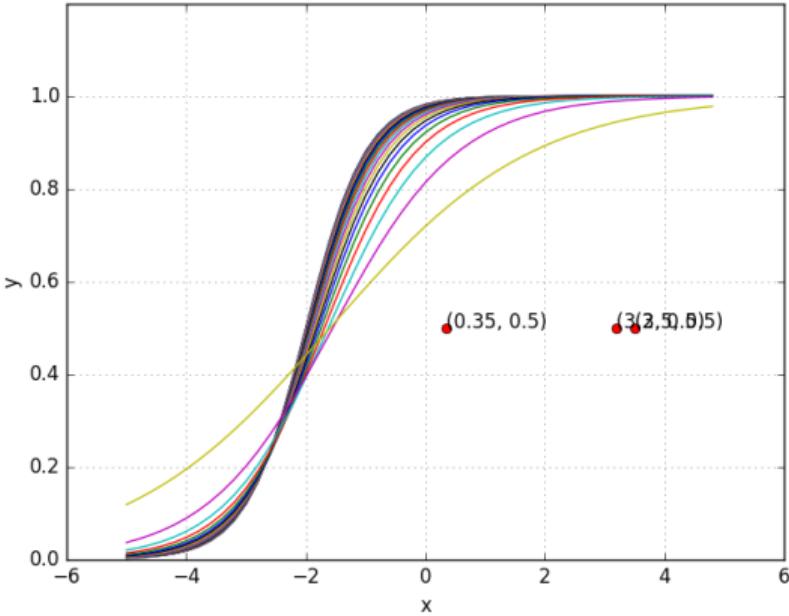
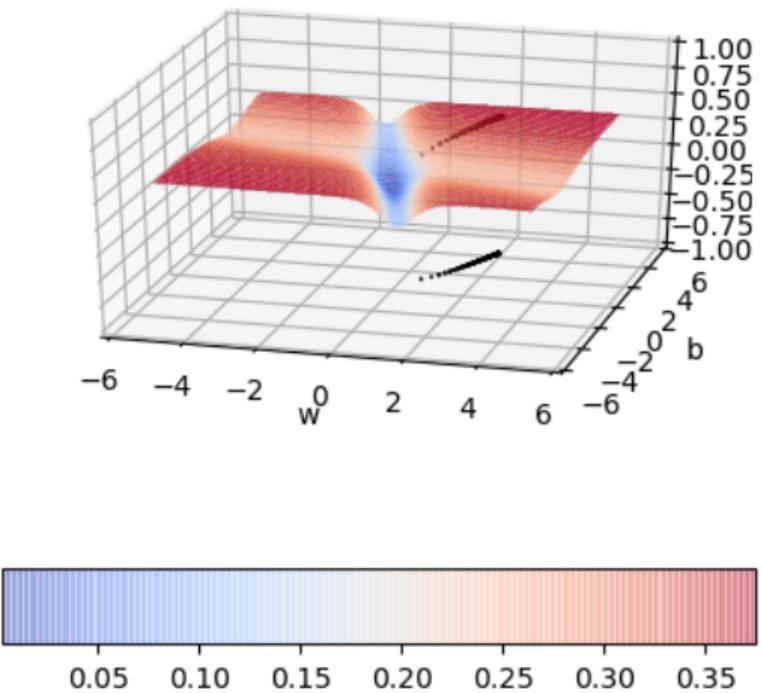


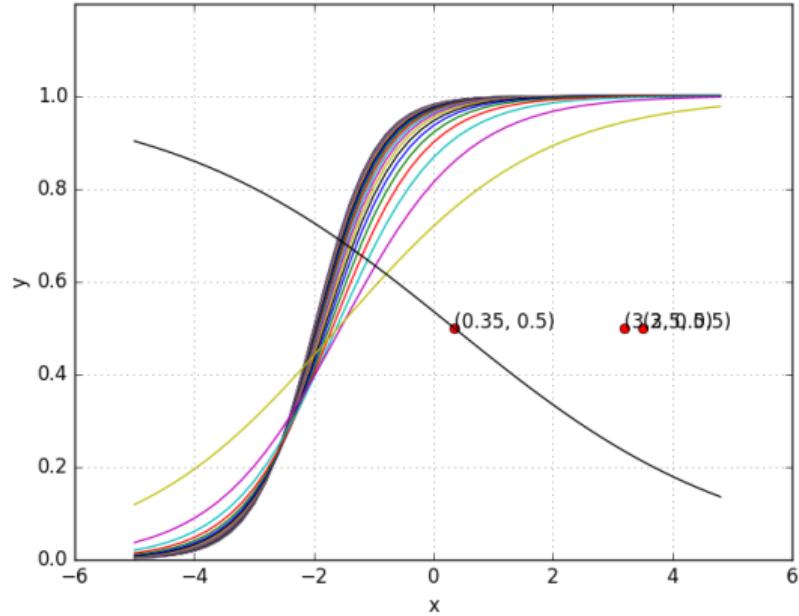
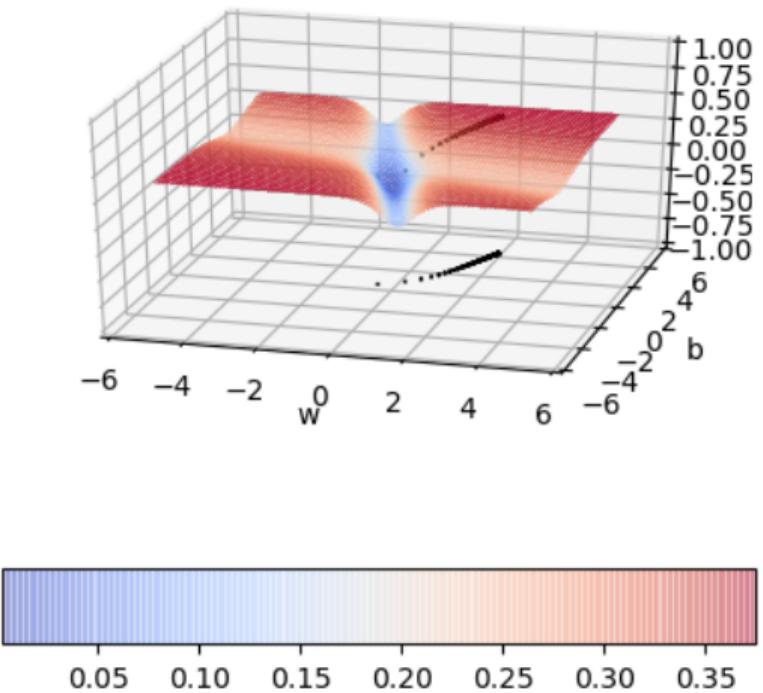


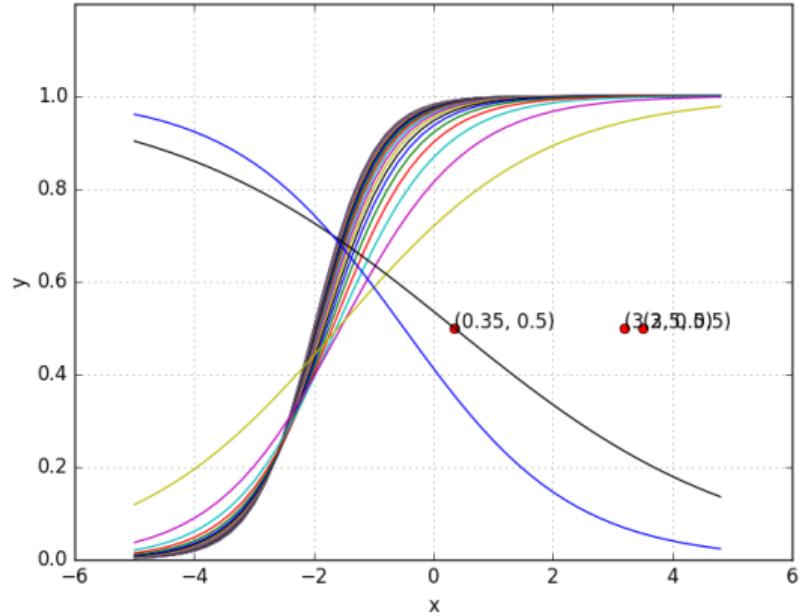
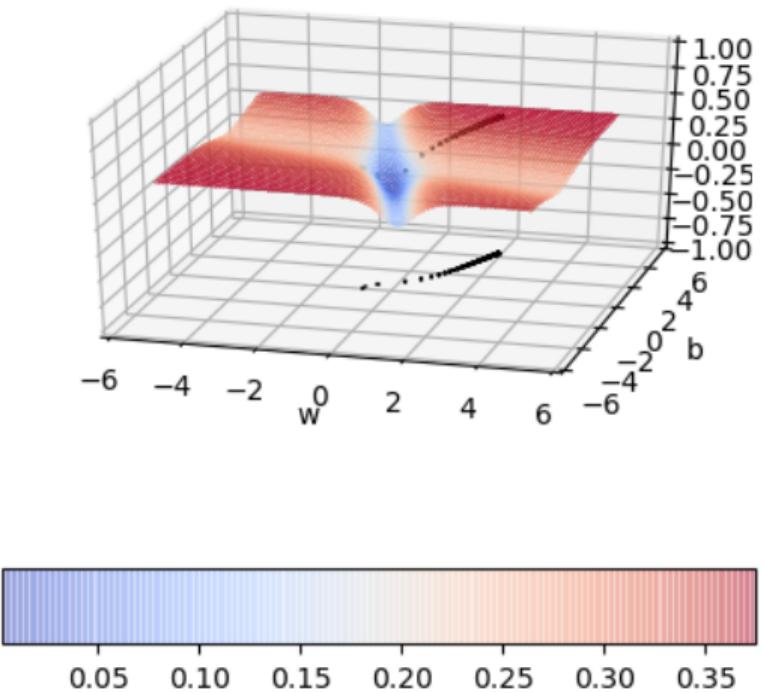


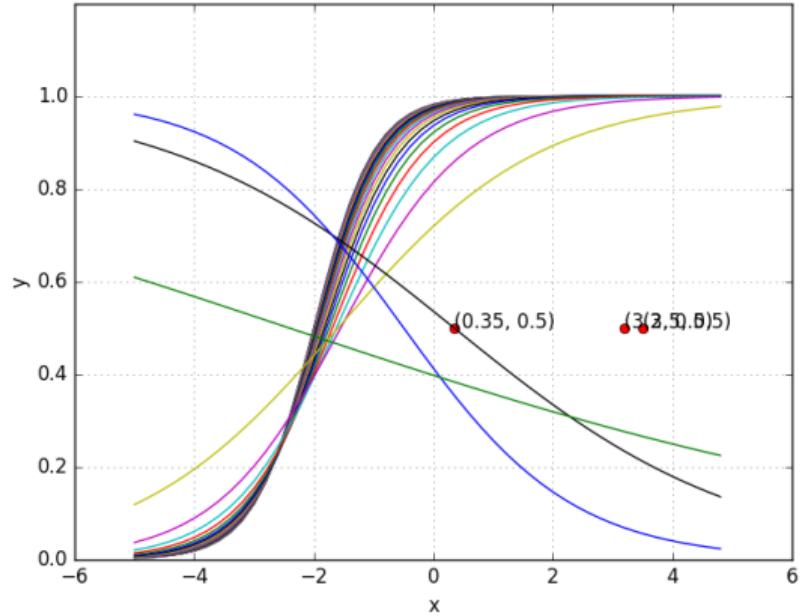
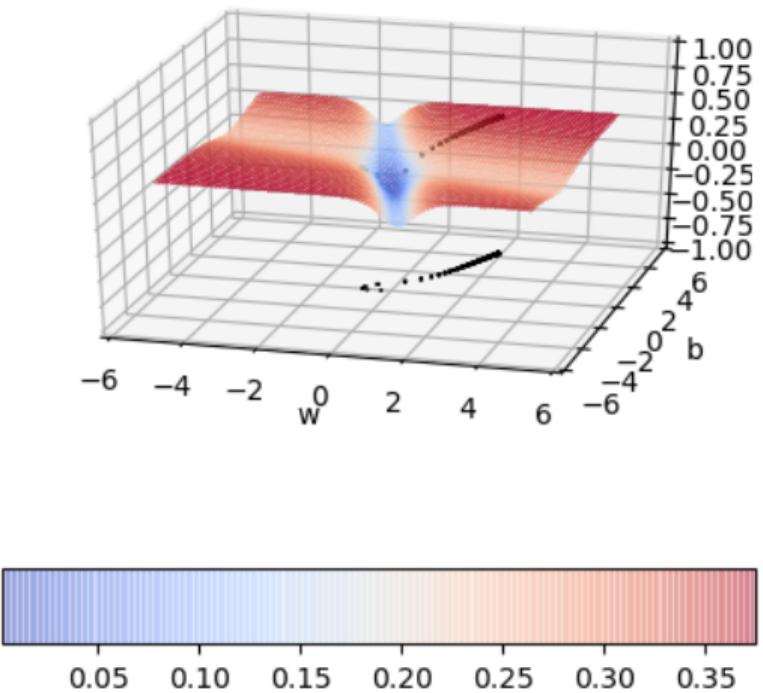


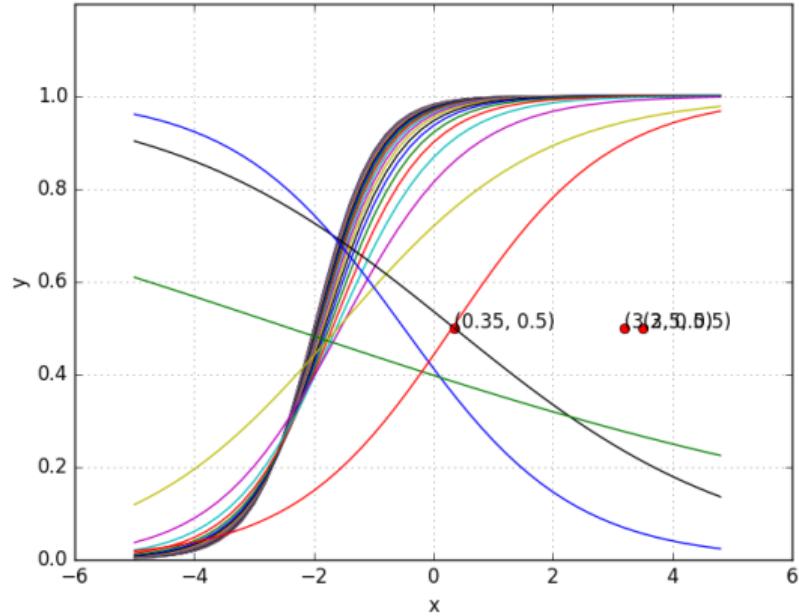
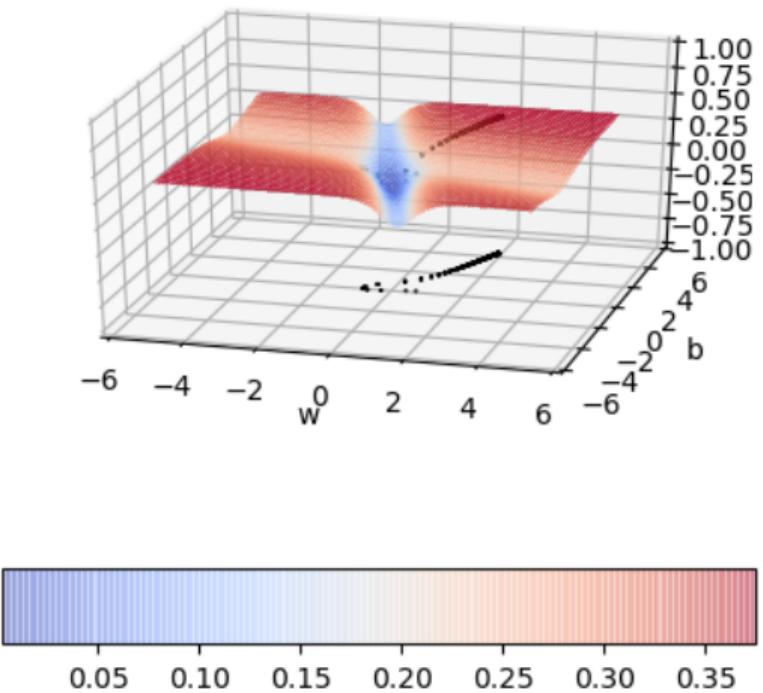


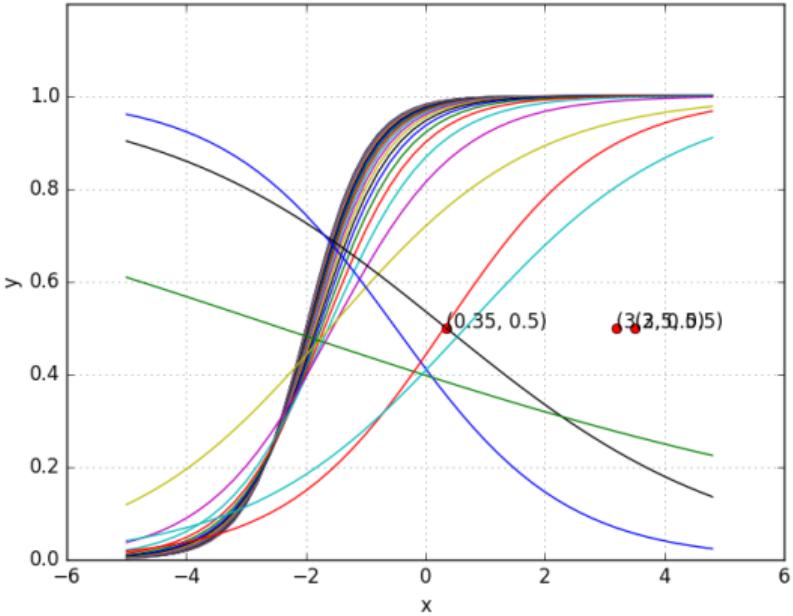
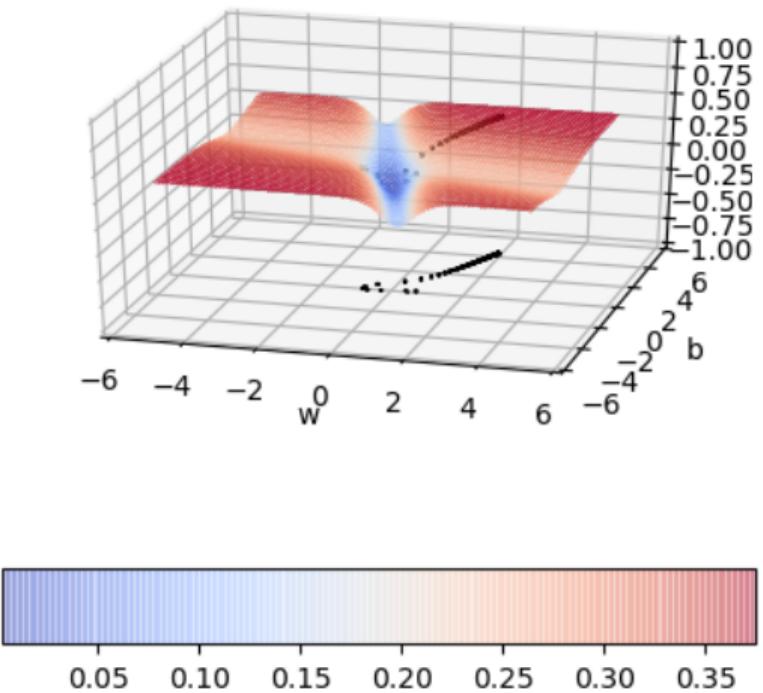


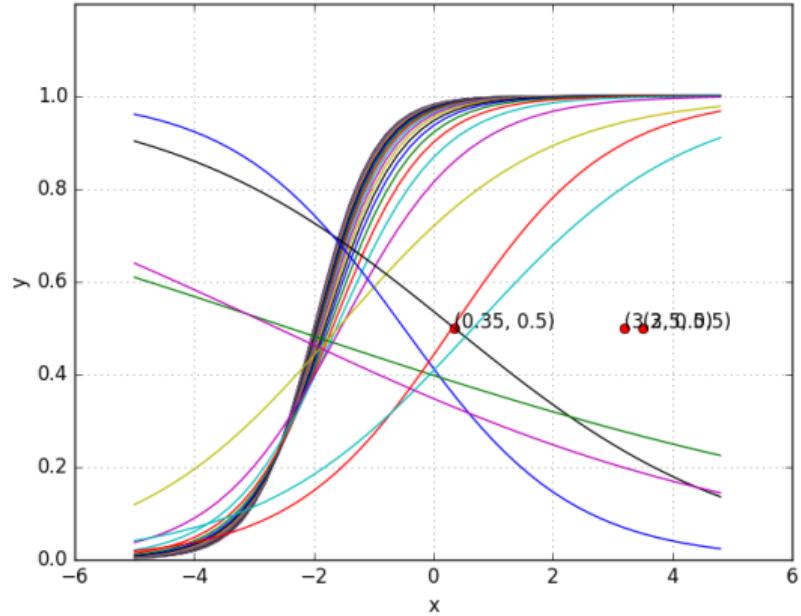
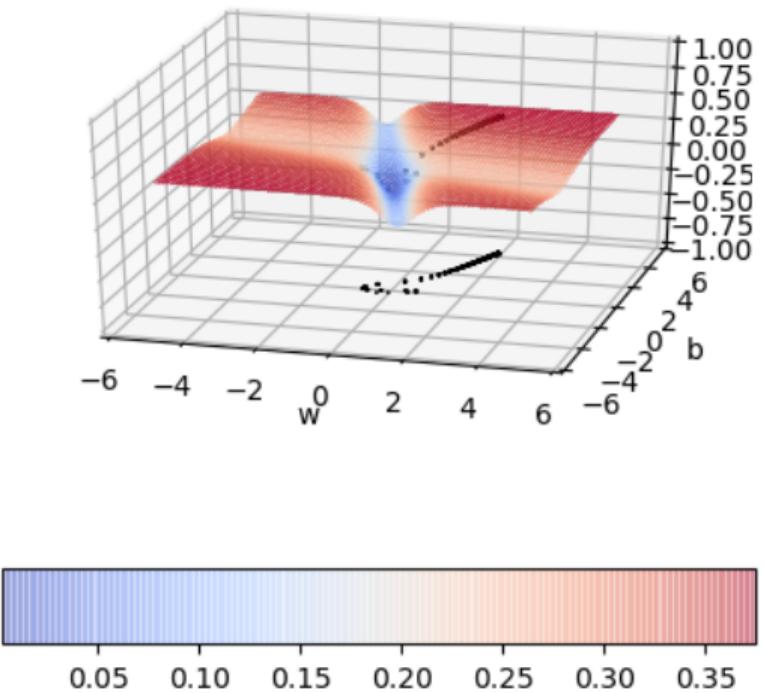


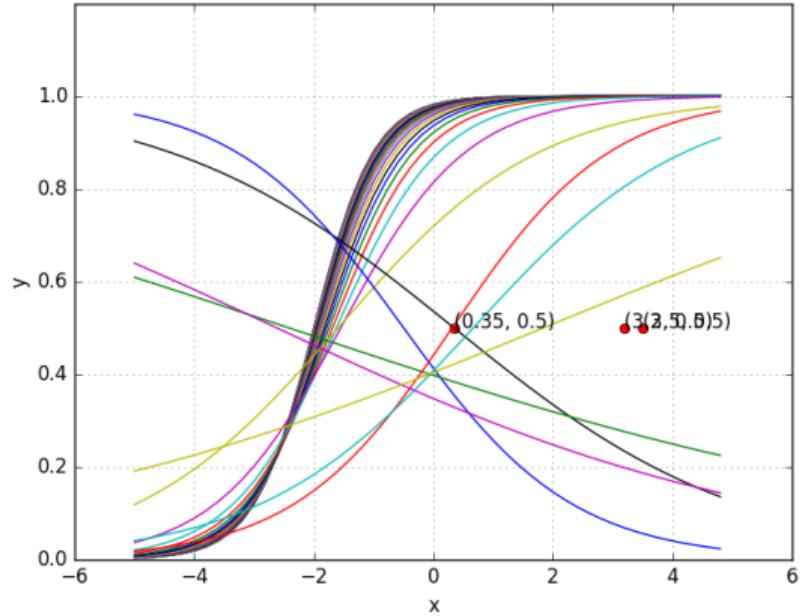
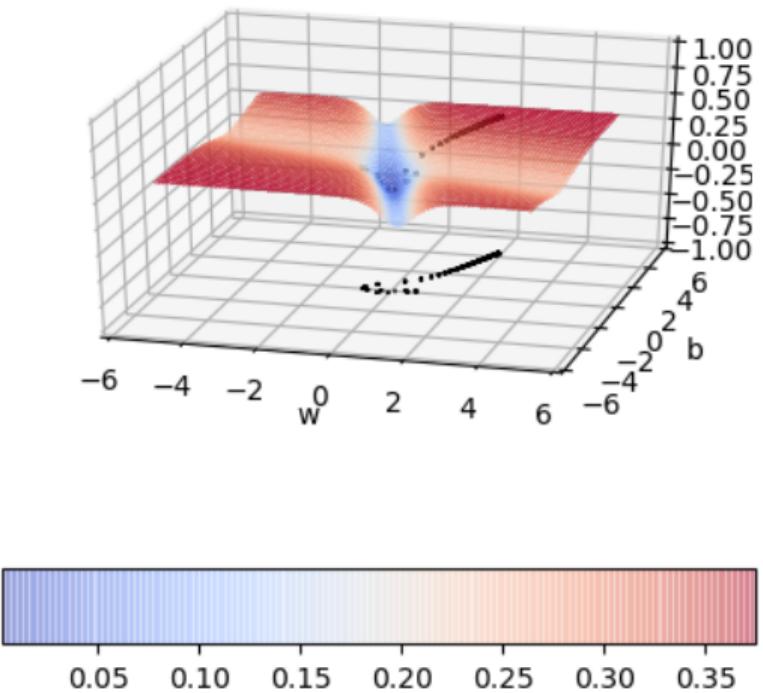


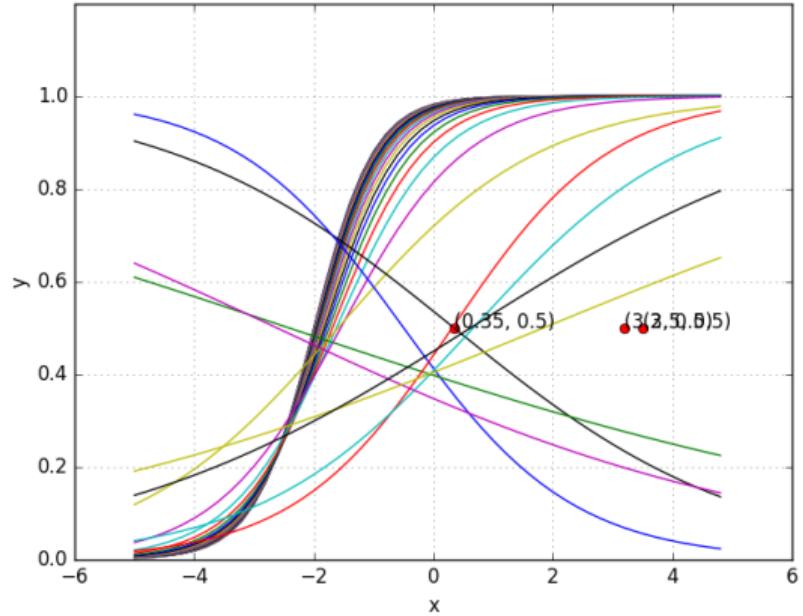
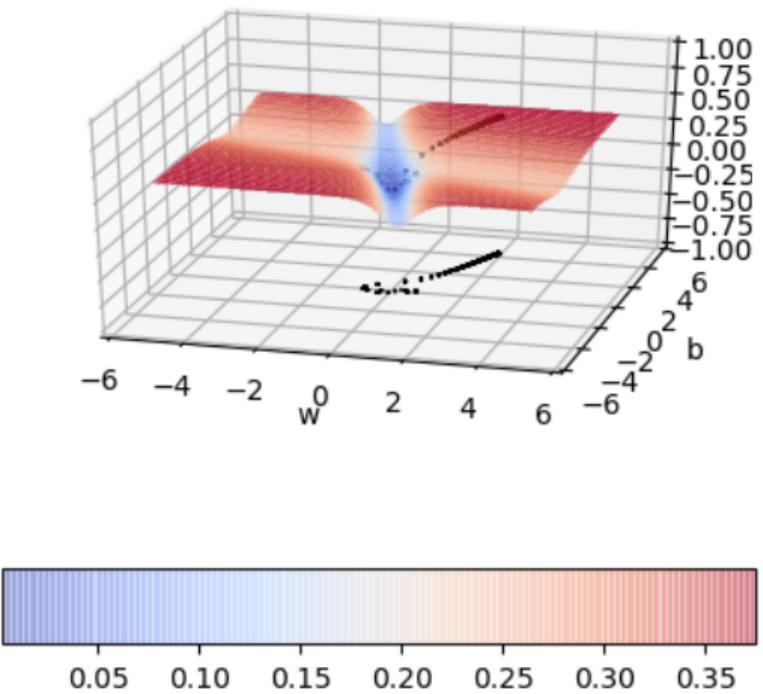


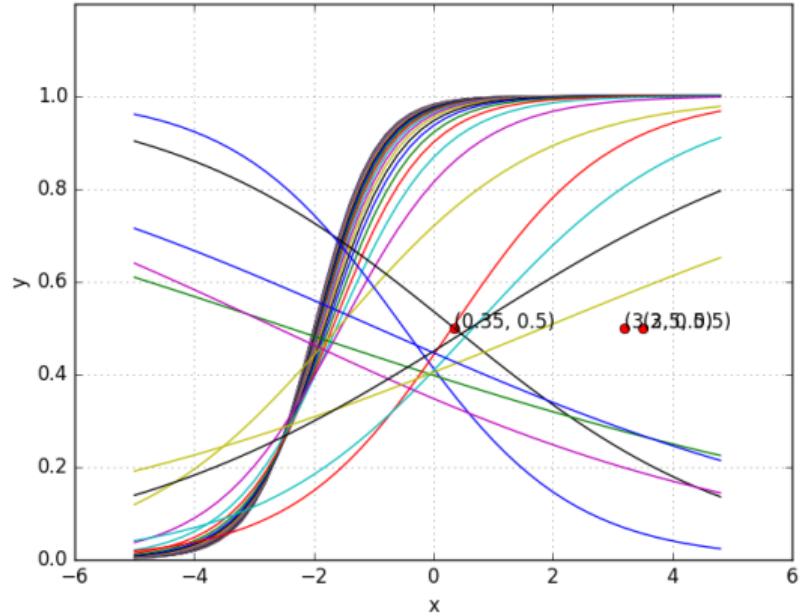
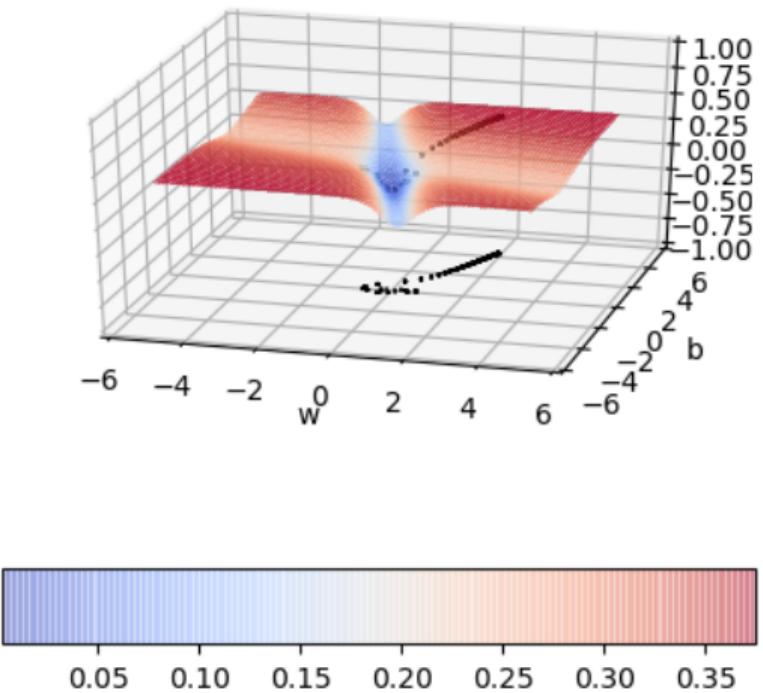


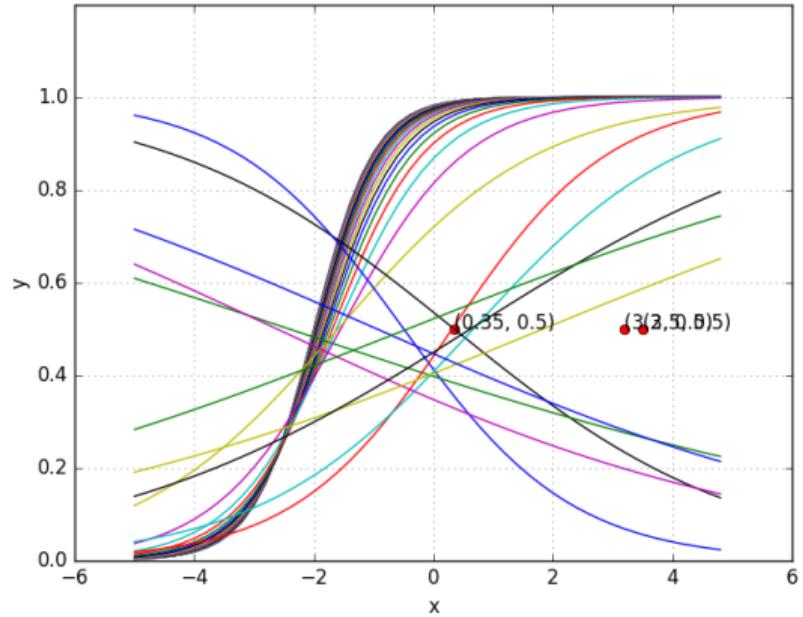
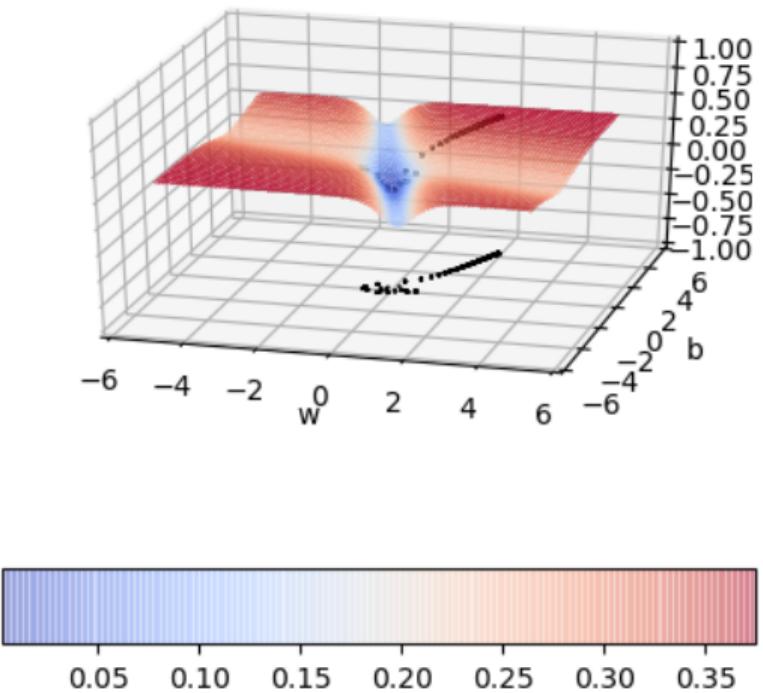


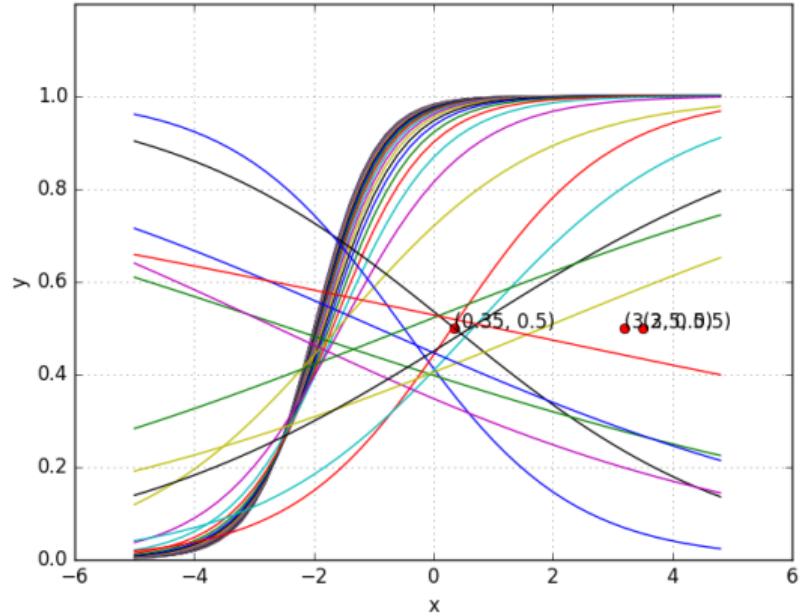
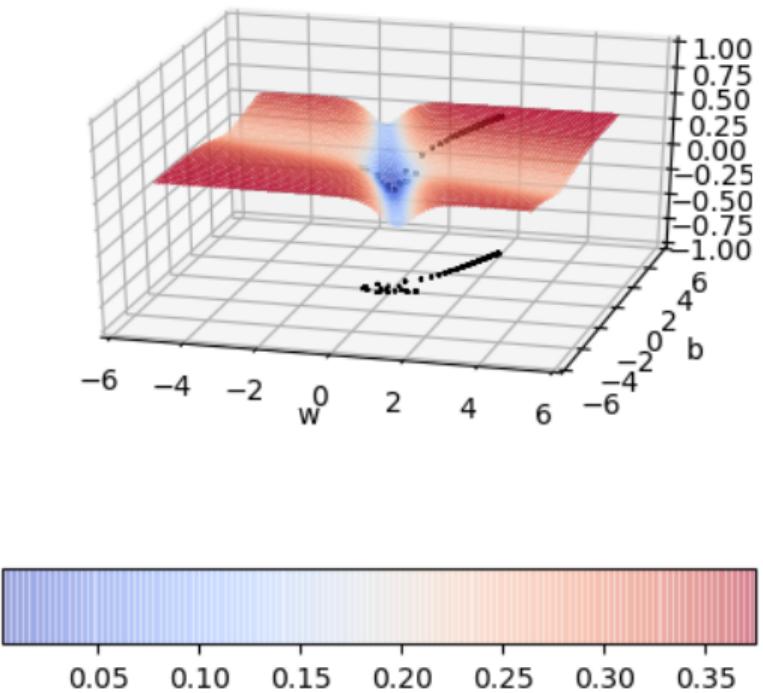


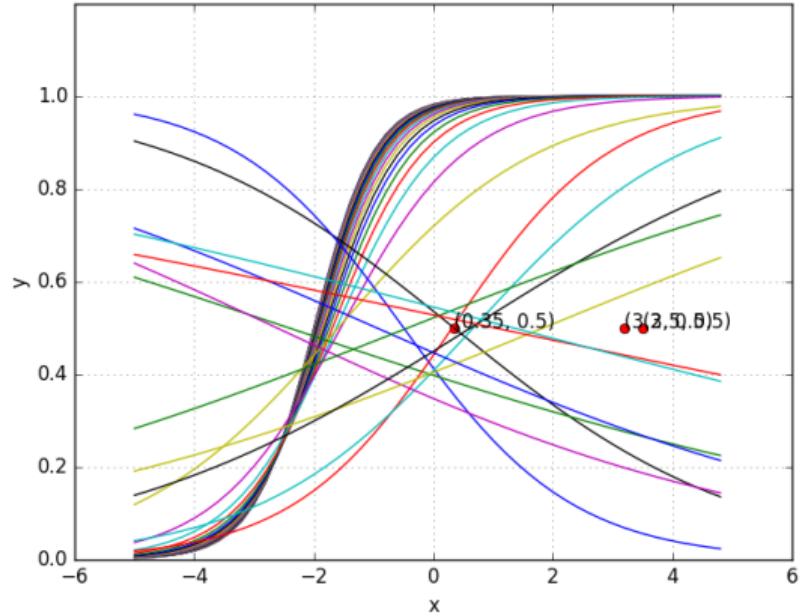
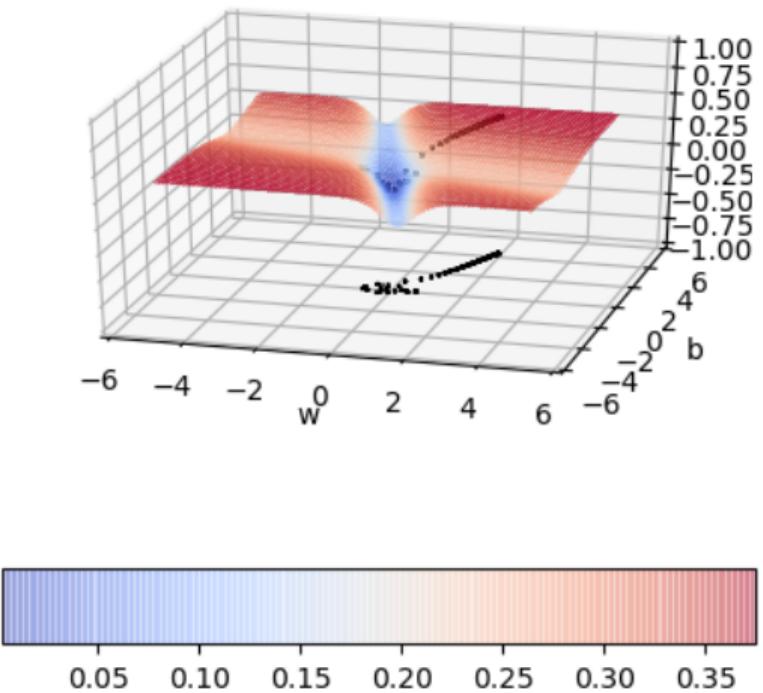


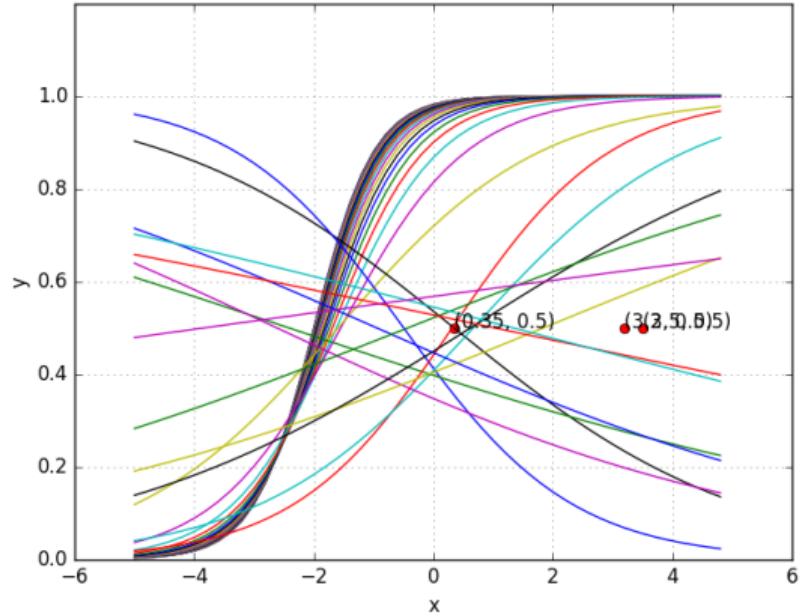
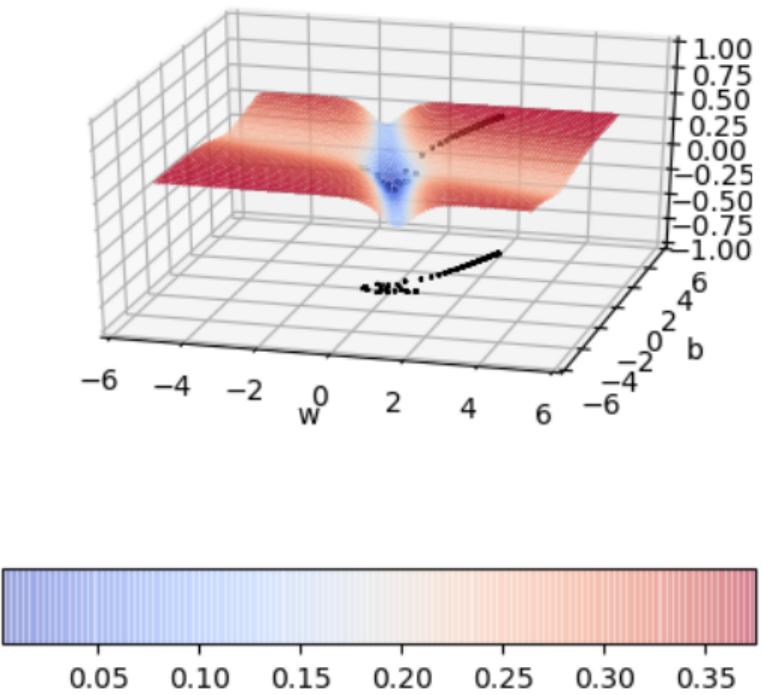


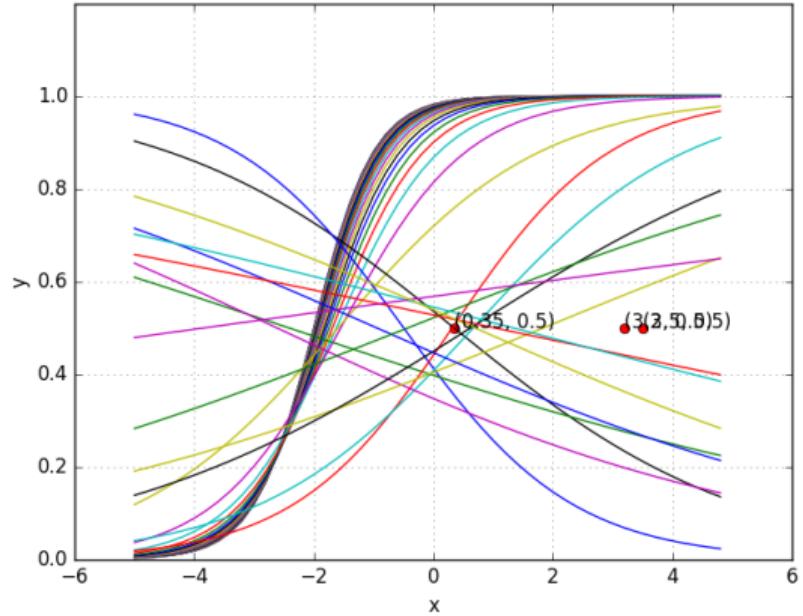
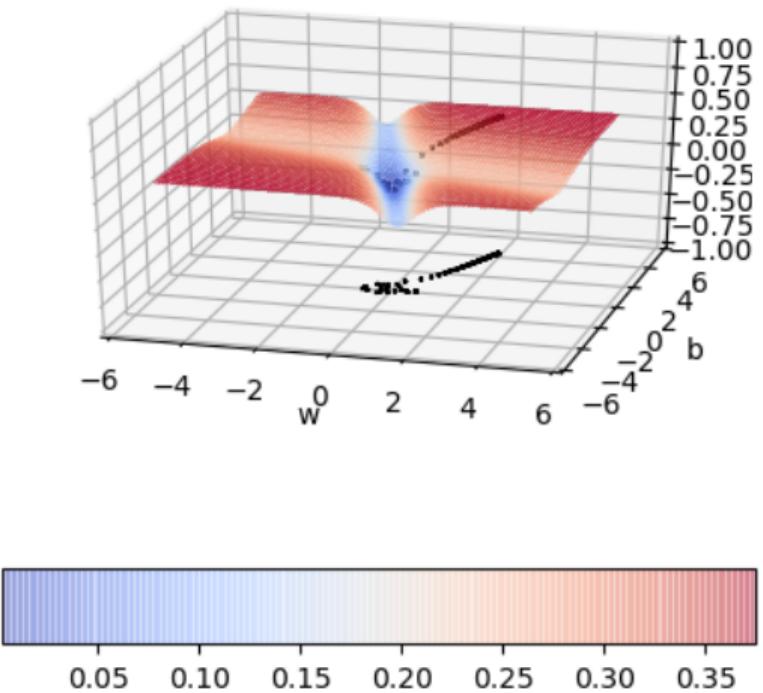


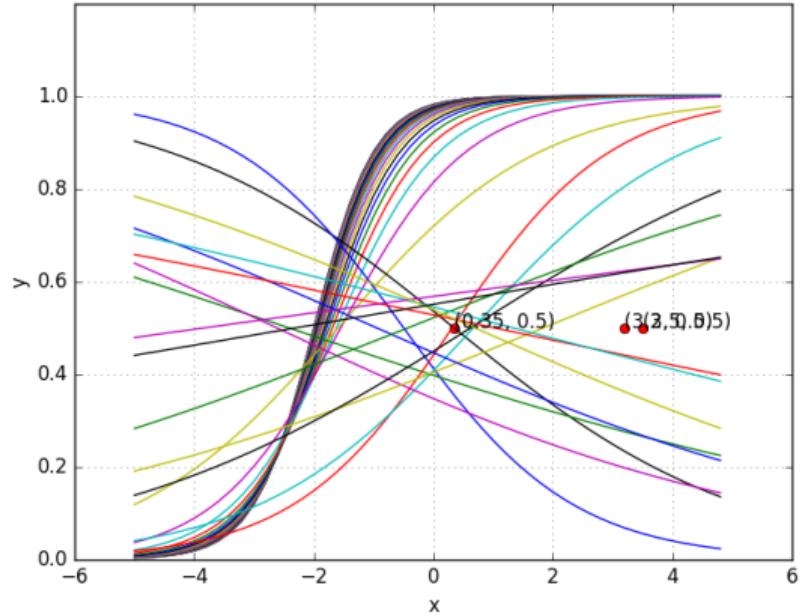
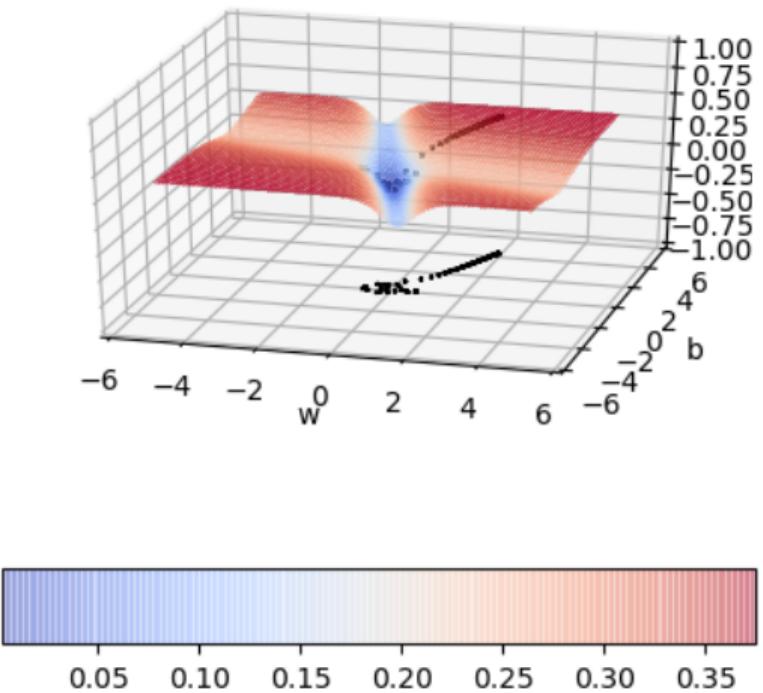


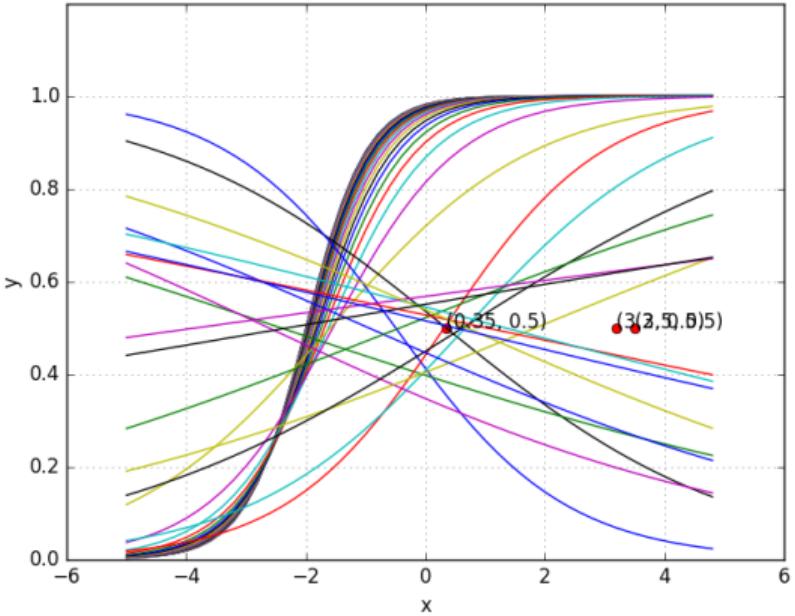
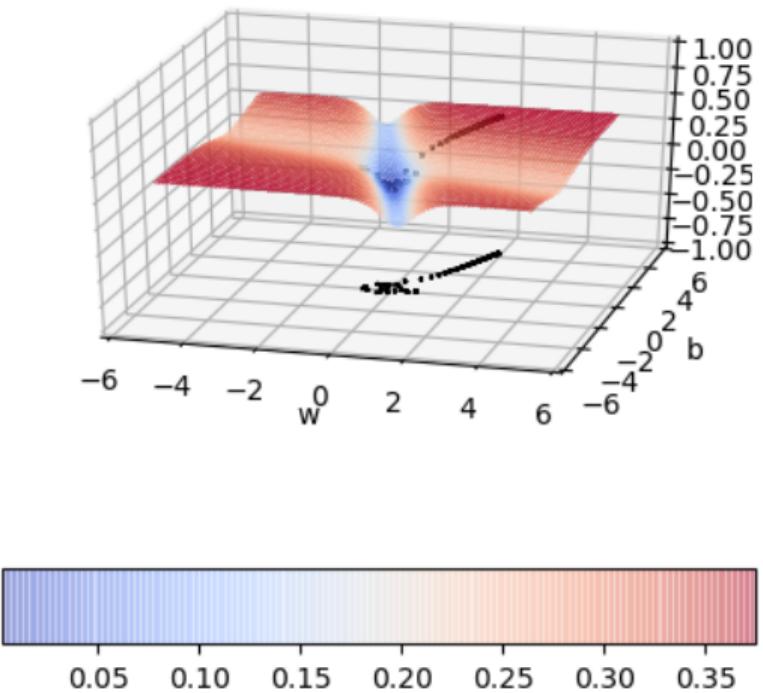


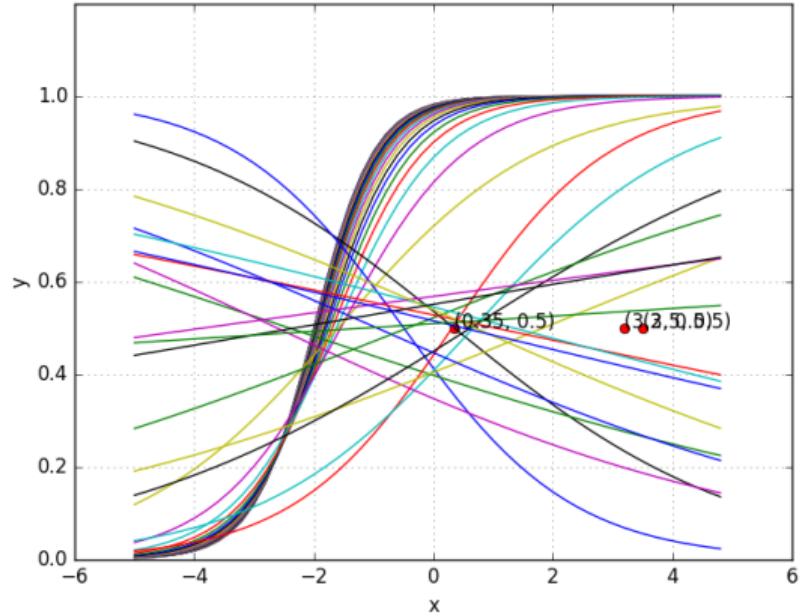
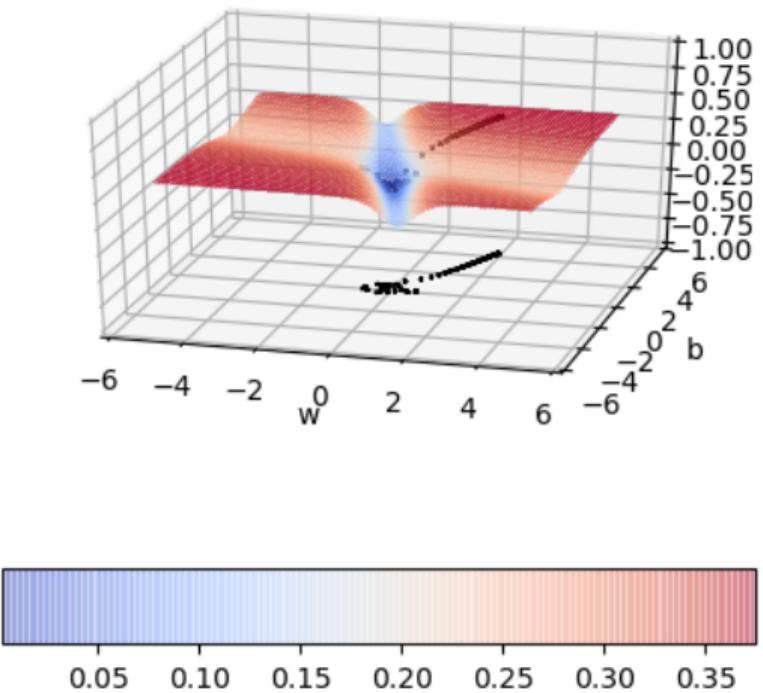


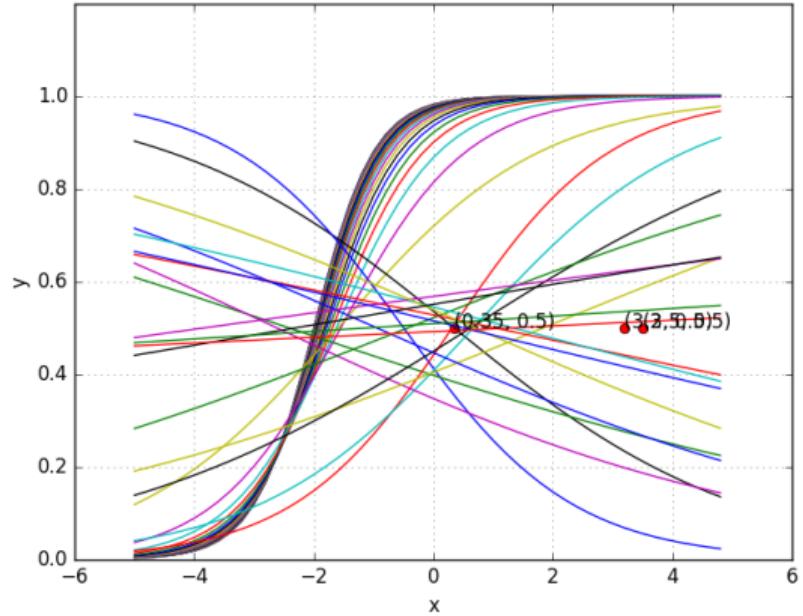
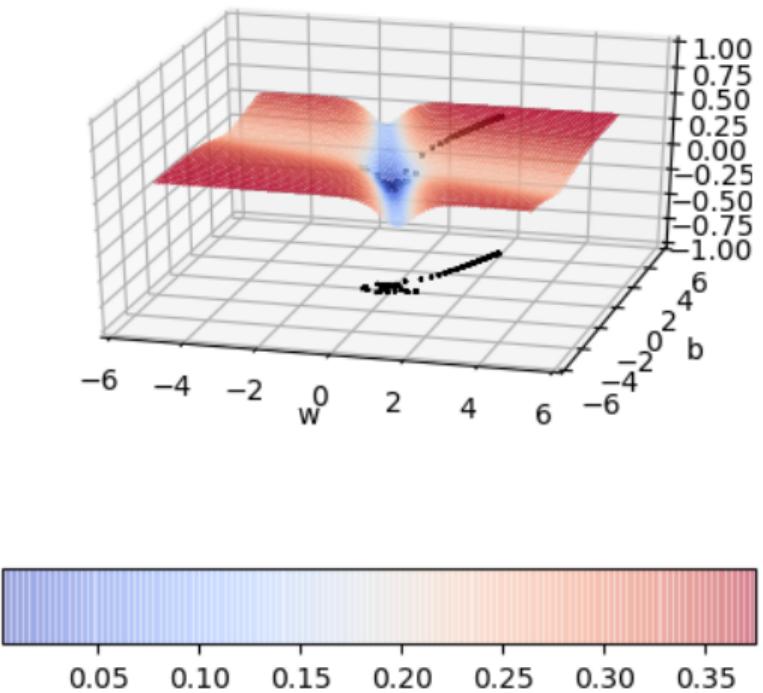


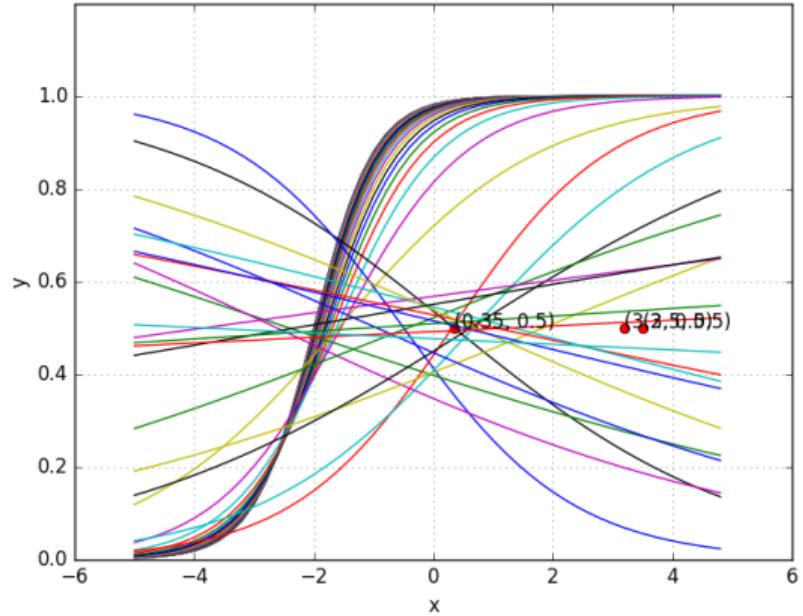
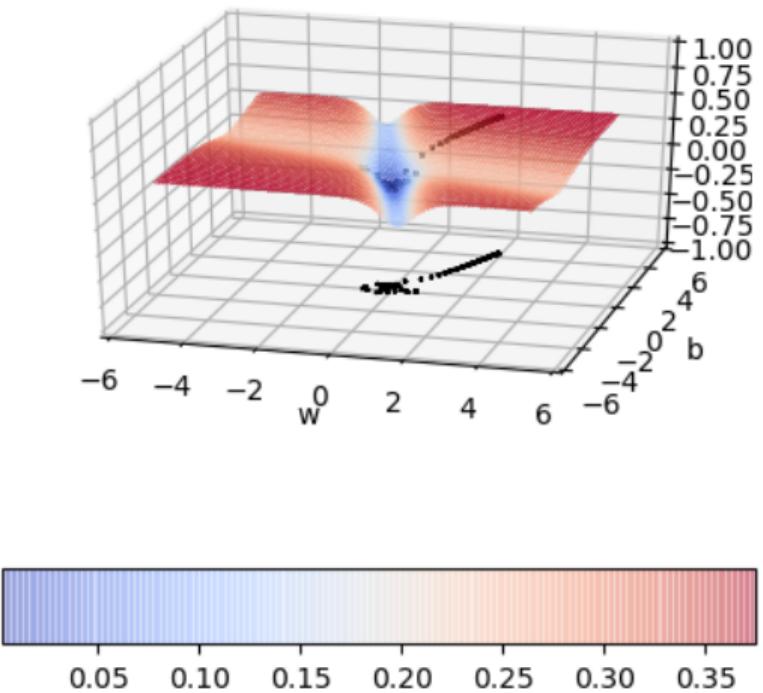


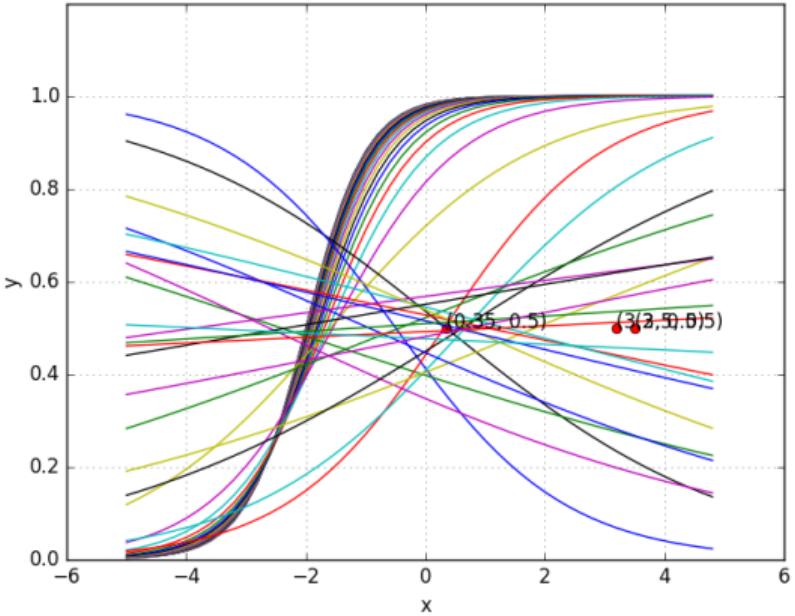
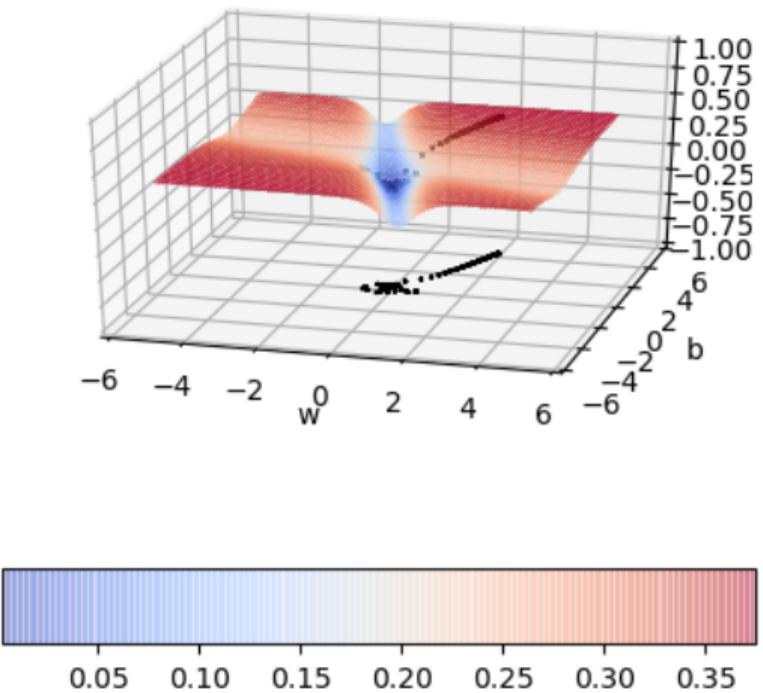


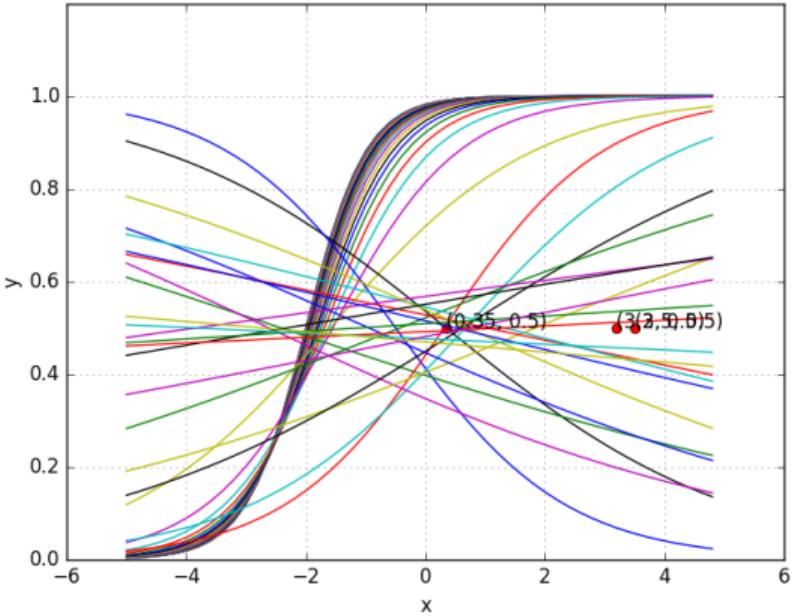
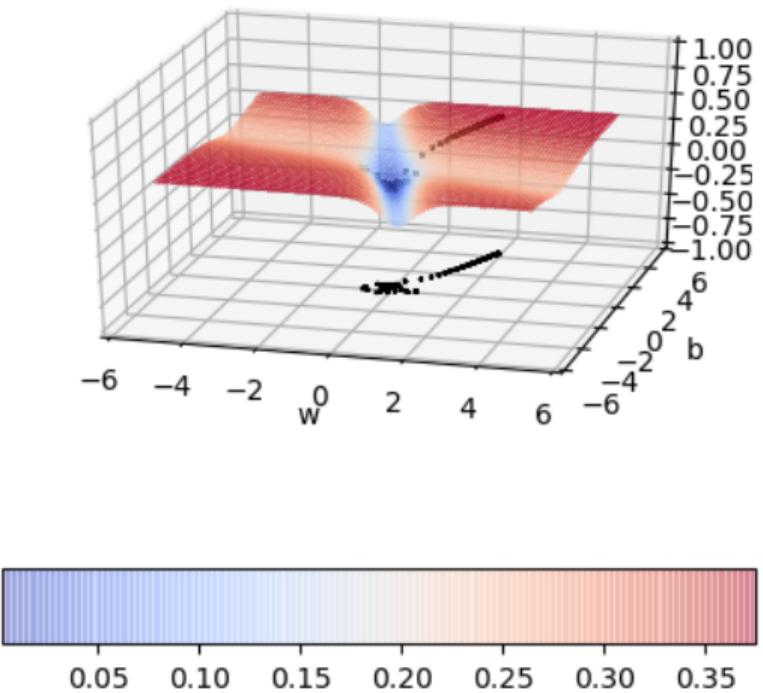














Nesterov Accelerated Gradient Descent



Question

- 能否进一步减少振荡?



Question

- 能否进一步减少振荡?
- 使用 Nesterov accelerated gradient Descent (NAGD)

Intuition

- 基于动量的梯度下降方法在计算当前时刻的更新量时 $update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$



Intuition

- 基于动量的梯度下降方法在计算当前时刻的更新量时 $update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$
- 当前时刻对参数的更新是先更新 $\gamma \cdot update_{t-1}$, 再更新 $\eta \nabla w_t$

Intuition

- 基于动量的梯度下降方法在计算当前时刻的更新量时 $update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$
- 当前时刻对参数的更新是先更新 $\gamma \cdot update_{t-1}$, 再更新 $\eta \nabla w_t$
- 为什么不在第一次更新之后的参数 w ($w_{look_ahead} = w_t + \gamma \cdot update_{t-1}$) 处计算梯度 (∇w_{look_ahead}) , 然后进行第二次更新 ?

Intuition

- 基于动量的梯度下降方法在计算当前时刻的更新量时 $update_t = \gamma \cdot update_{t-1} - \eta \nabla w_t$
- 当前时刻对参数的更新是先更新 $\gamma \cdot update_{t-1}$, 再更新 $\eta \nabla w_t$
- 为什么不在第一次更新之后的参数 w ($w_{look_ahead} = w_t + \gamma \cdot update_{t-1}$) 处计算梯度 (∇w_{look_ahead}) , 然后进行第二次更新 ?

NAGD 的更新规则

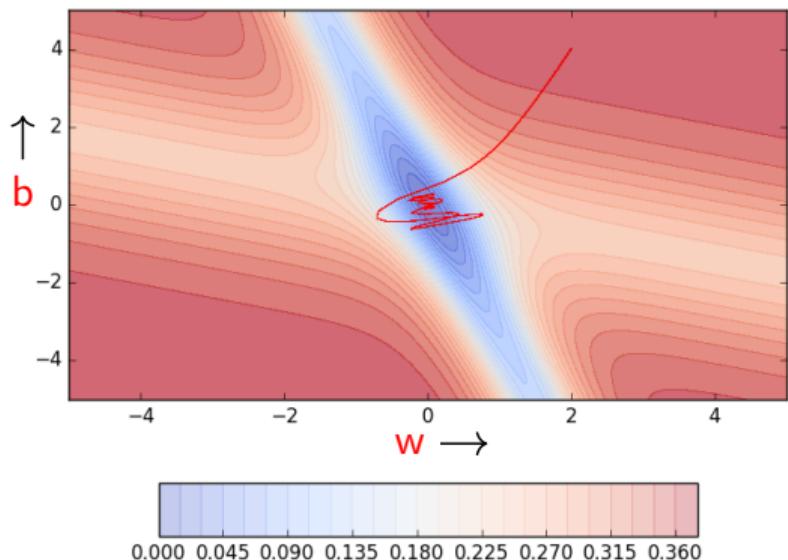
$$w_{look_ahead} = w_t + \gamma \cdot update_{t-1}$$

$$update_t = \gamma \cdot update_{t-1} + \eta \nabla w_{look_ahead}$$

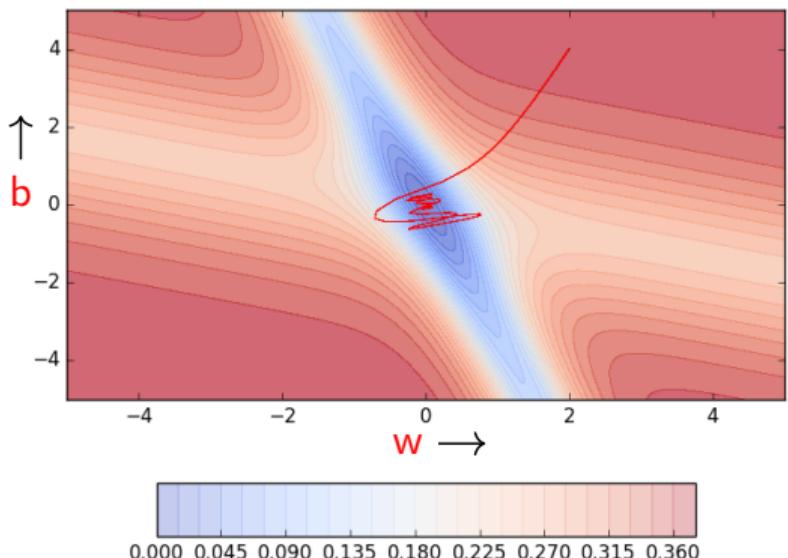
$$w_{t+1} = w_t + update_t$$

对 b_t 有相同的更新规则

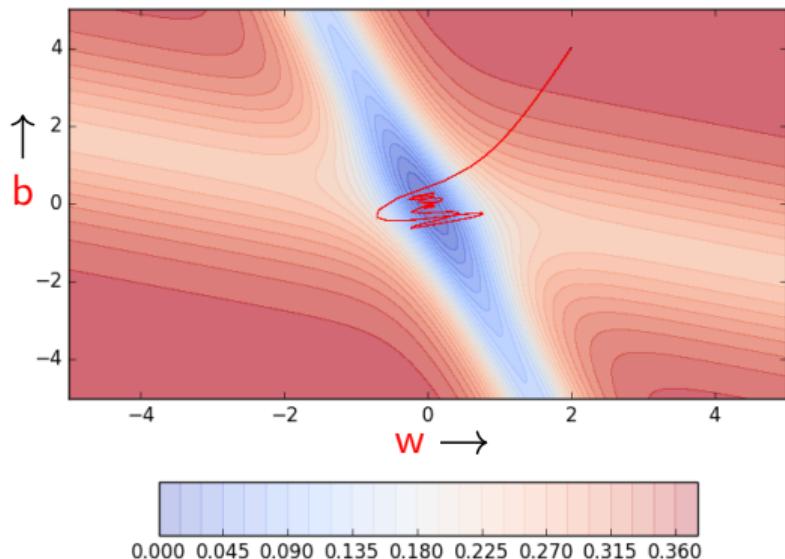
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



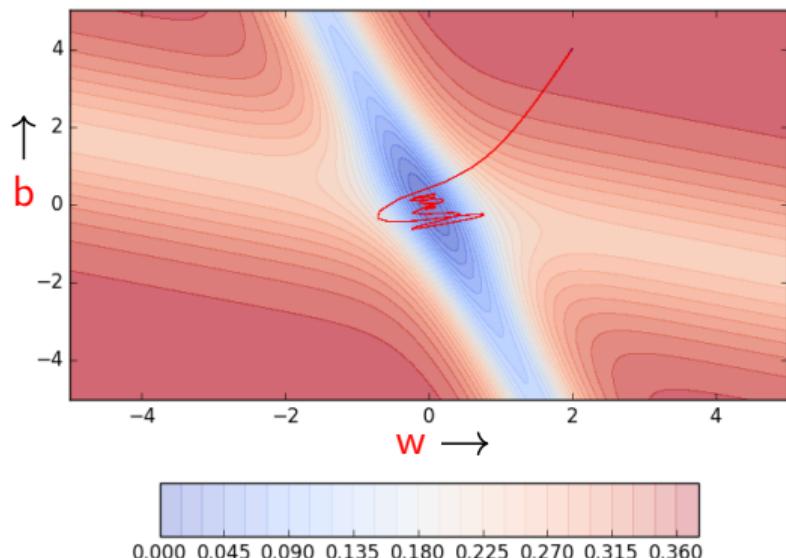
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



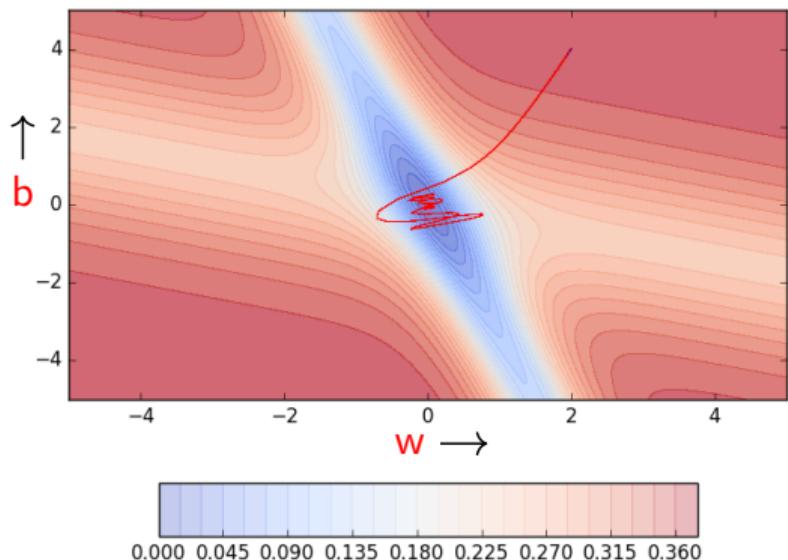
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



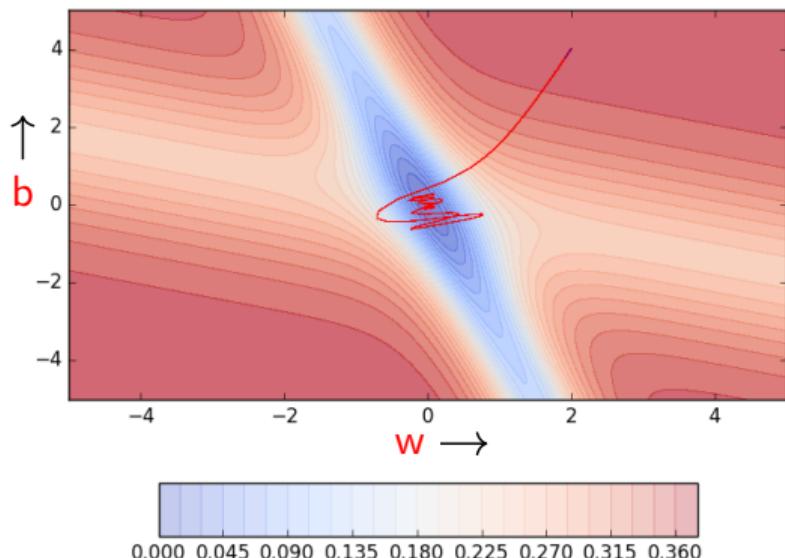
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



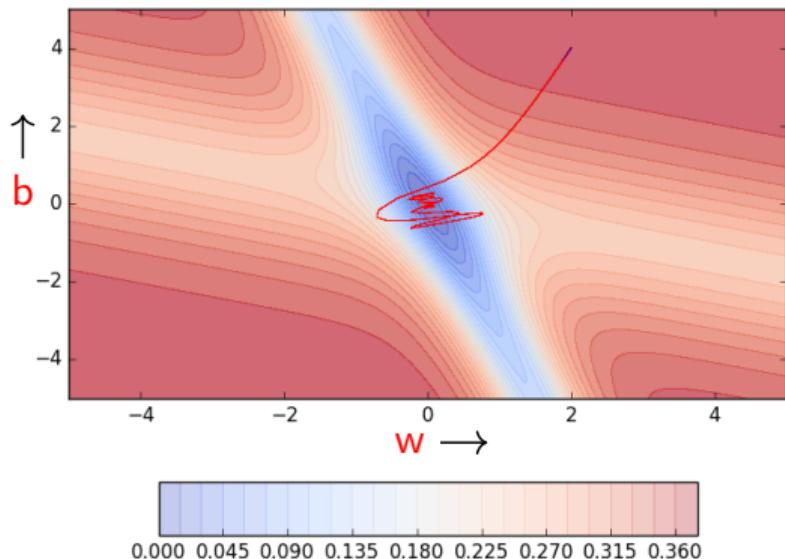
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



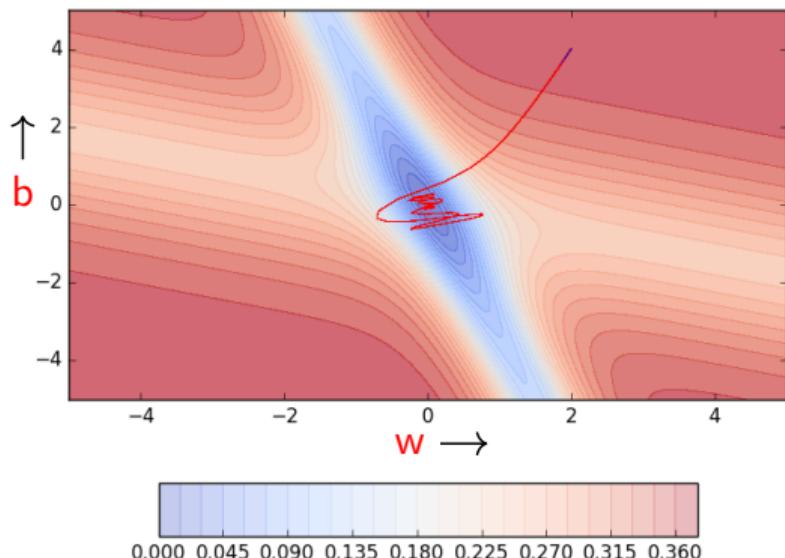
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



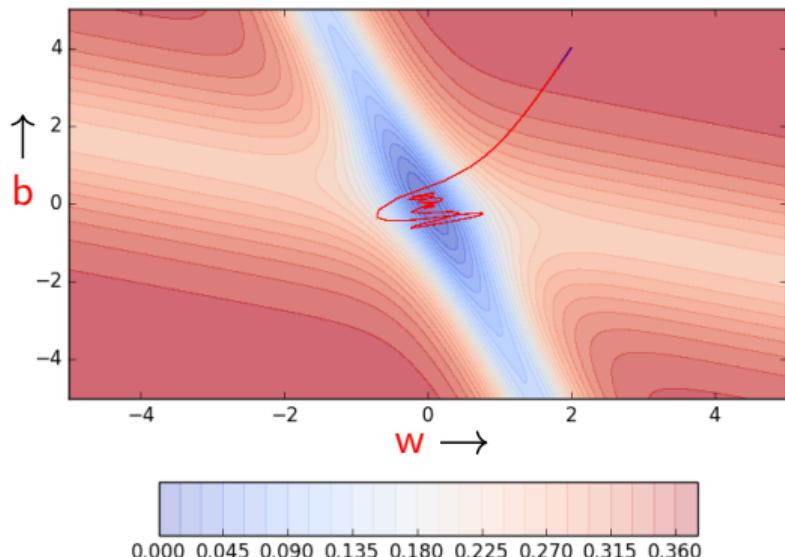
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



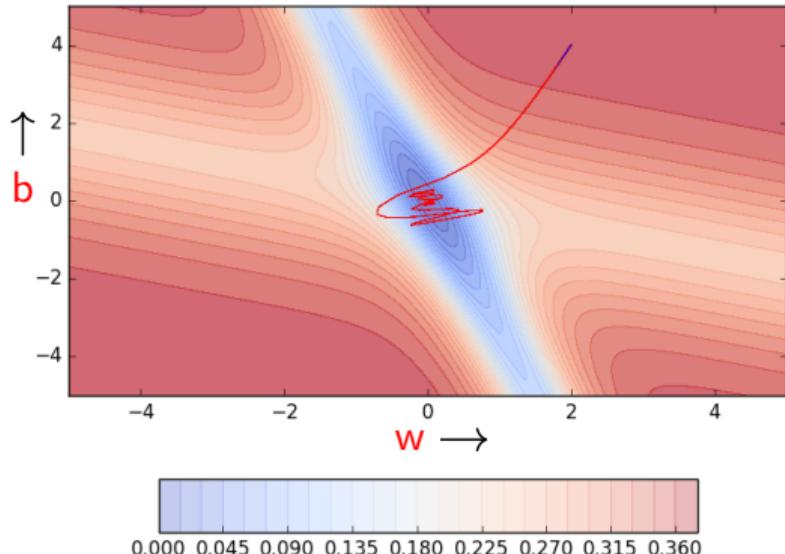
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



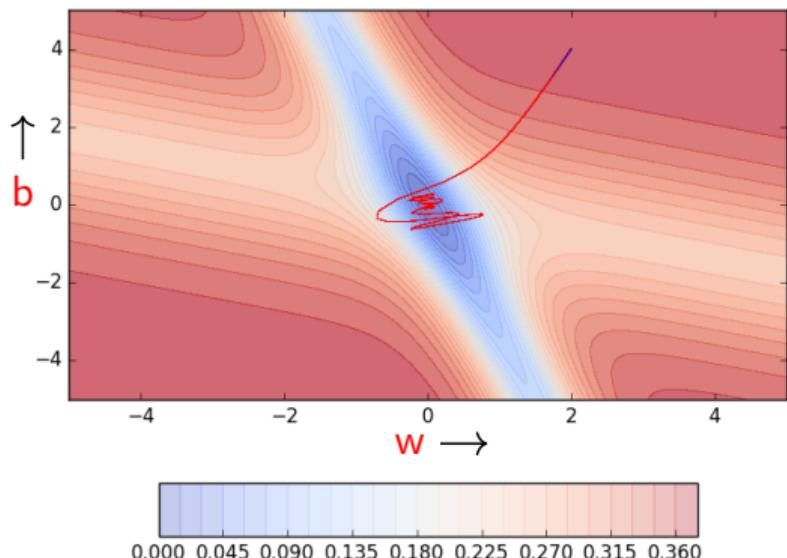
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



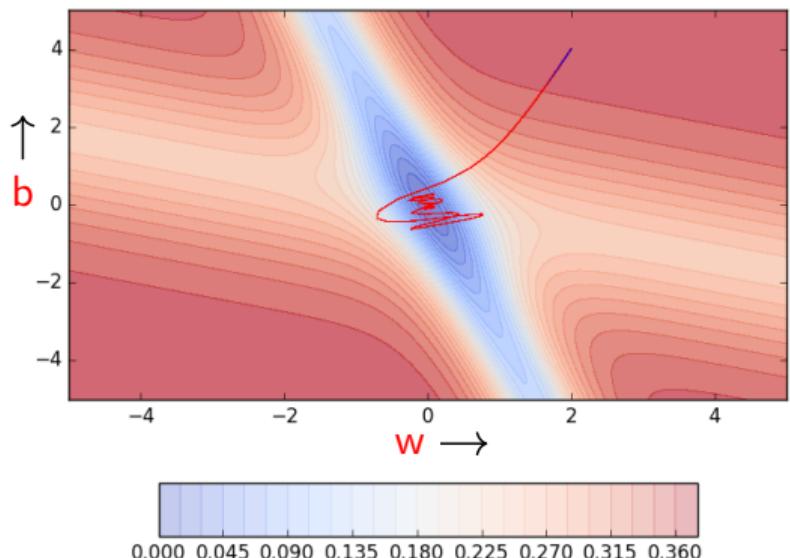
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



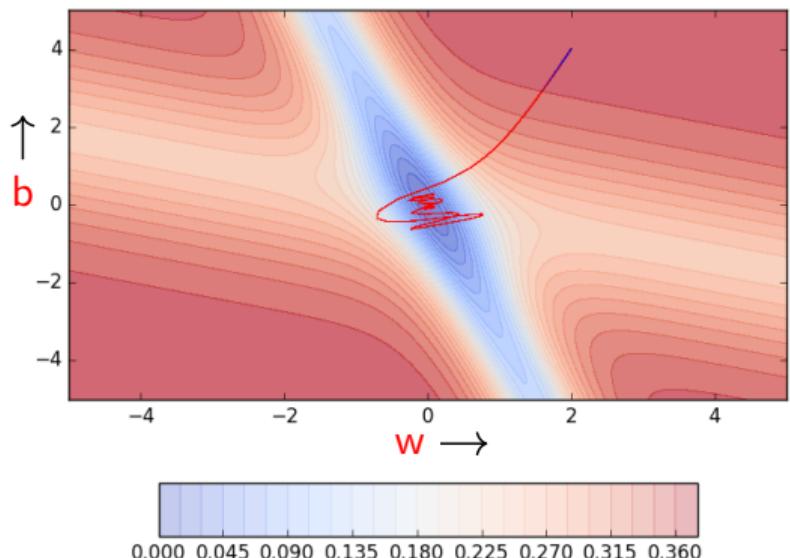
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



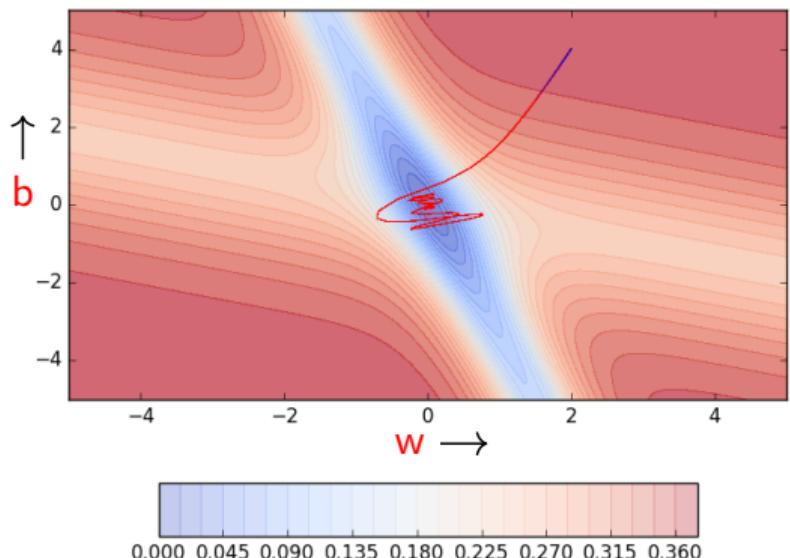
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

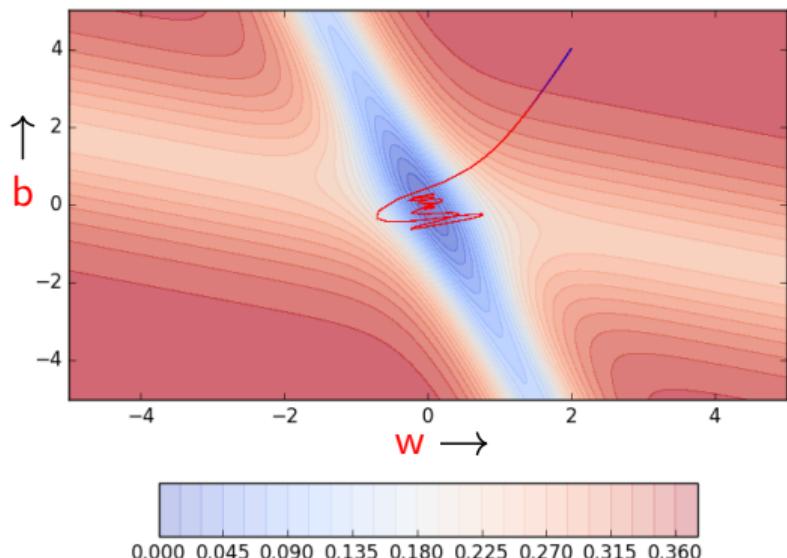


```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

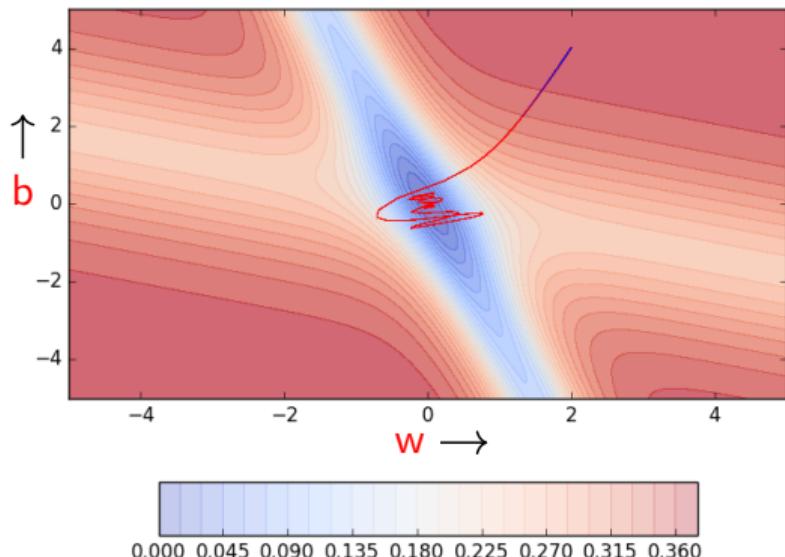




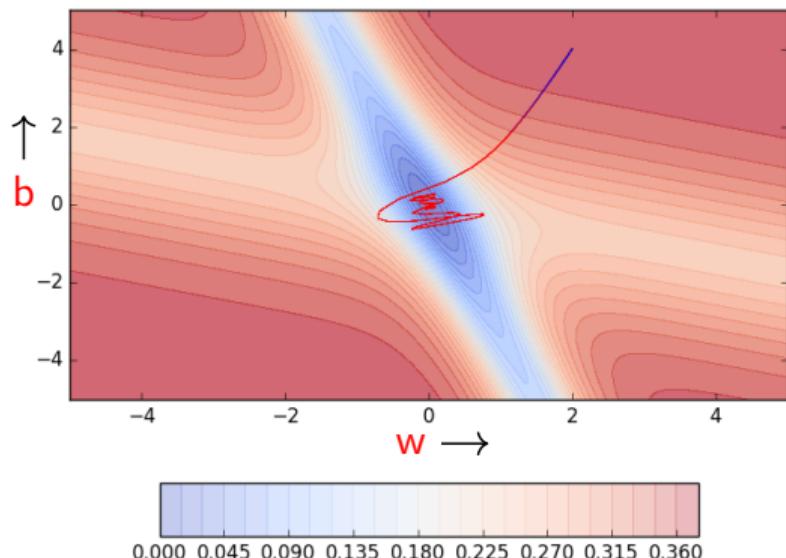
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



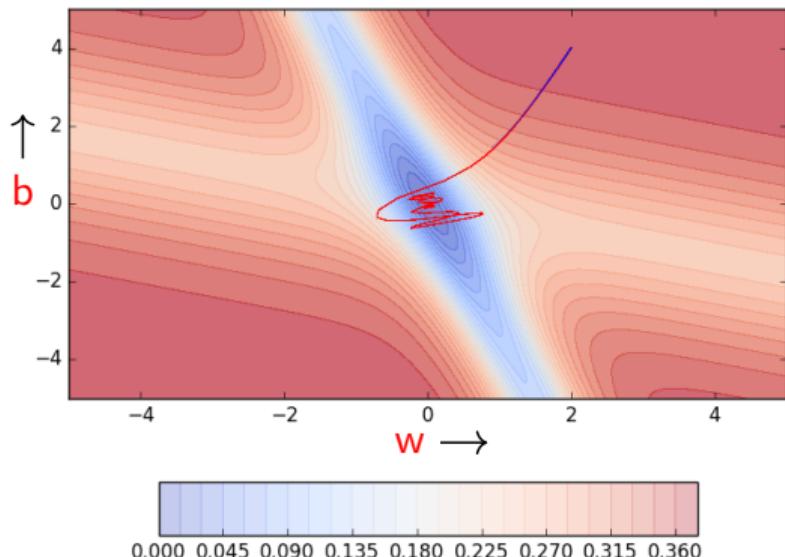
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



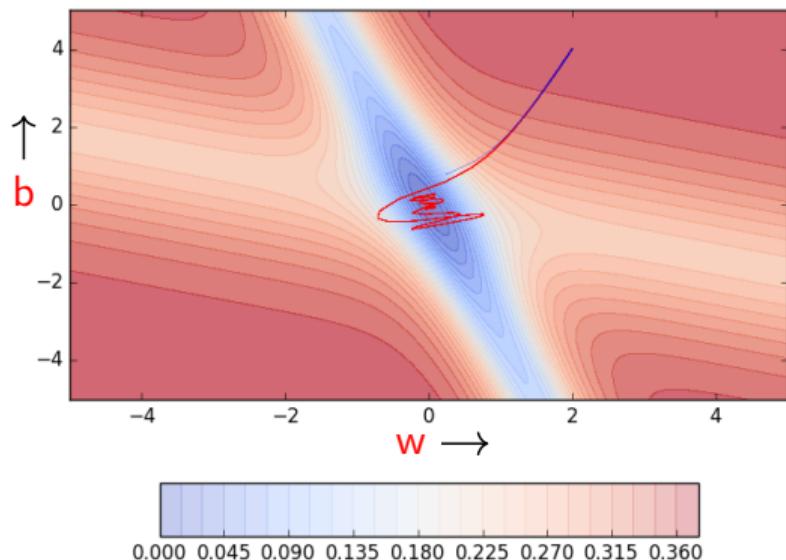
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



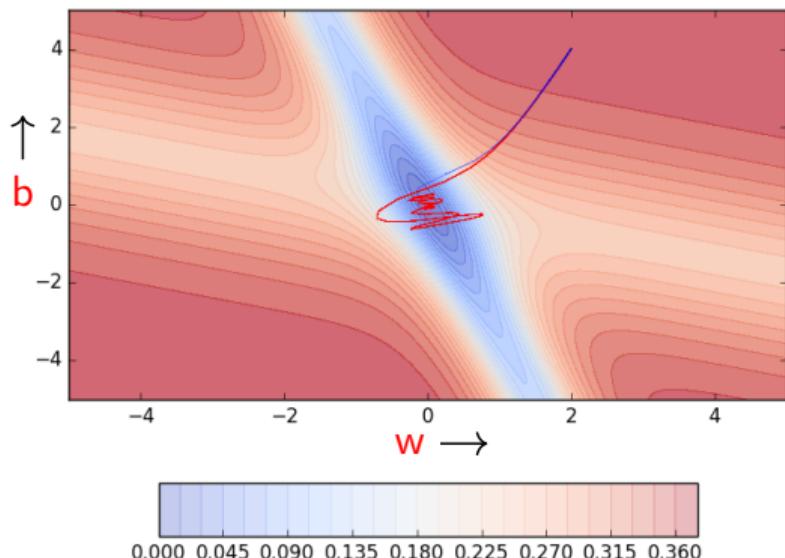
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

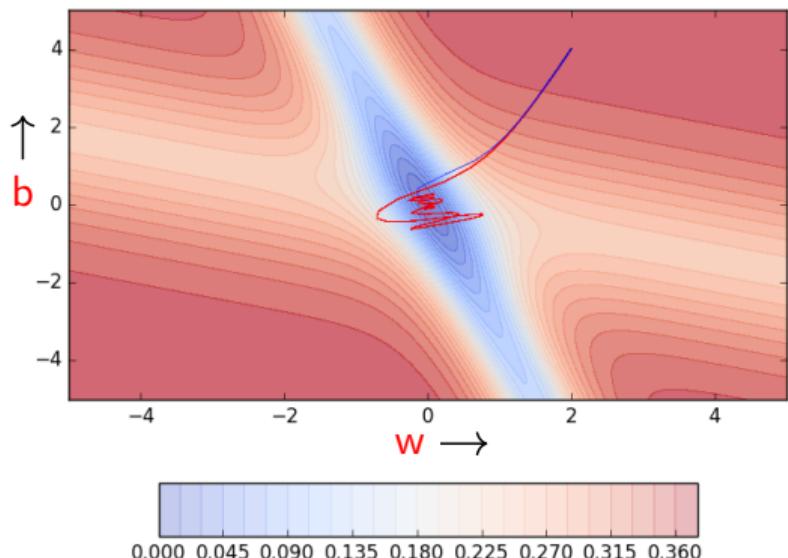


```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

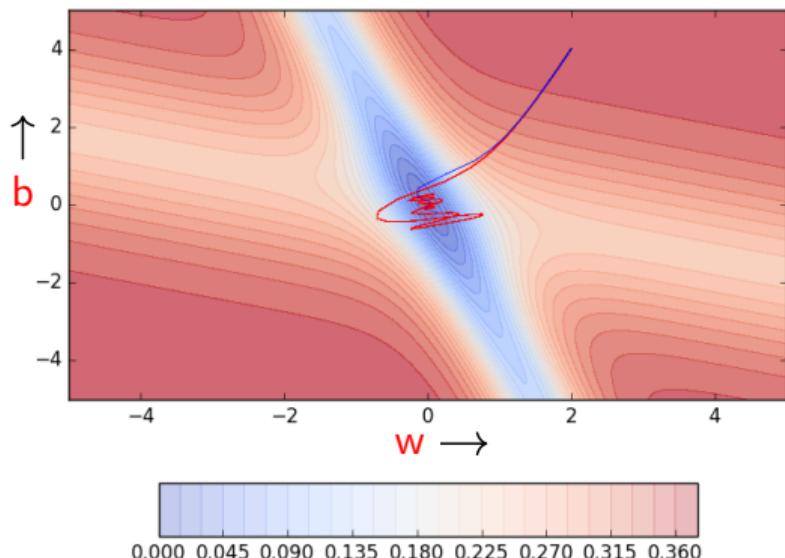




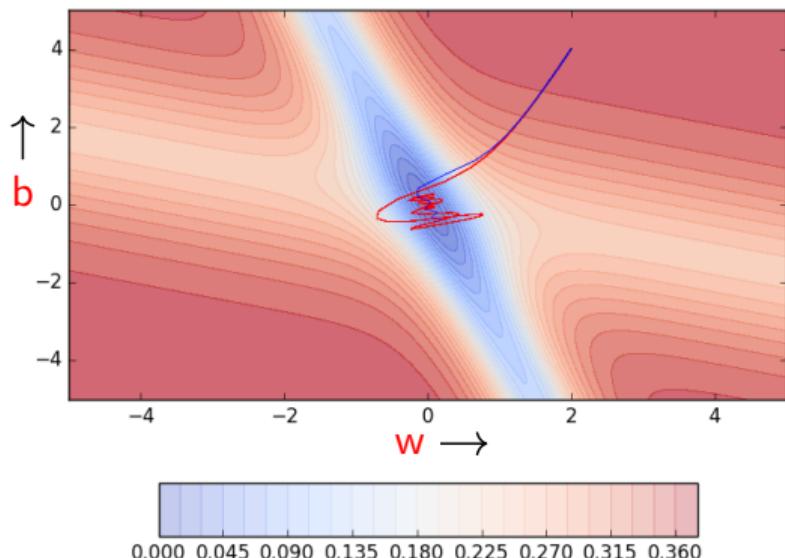
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



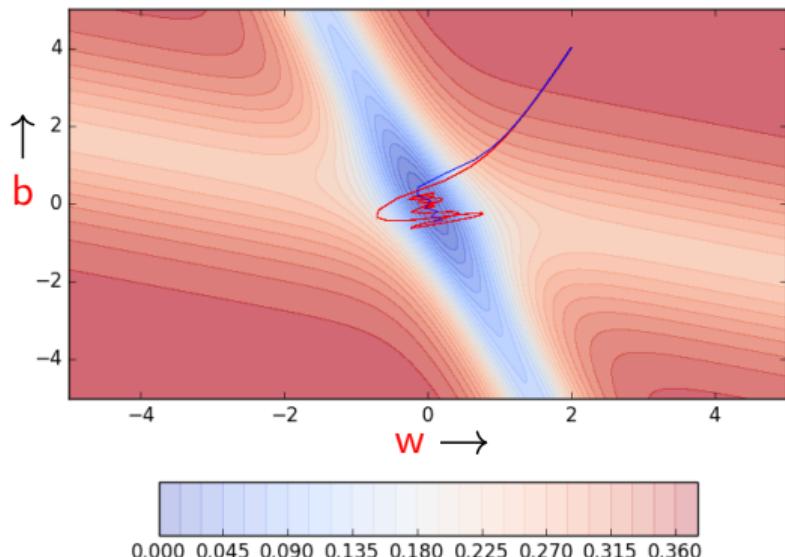
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



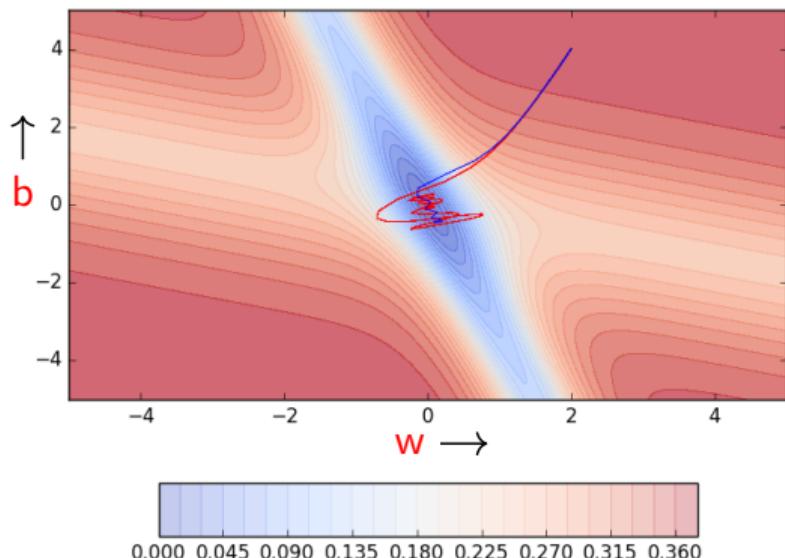
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



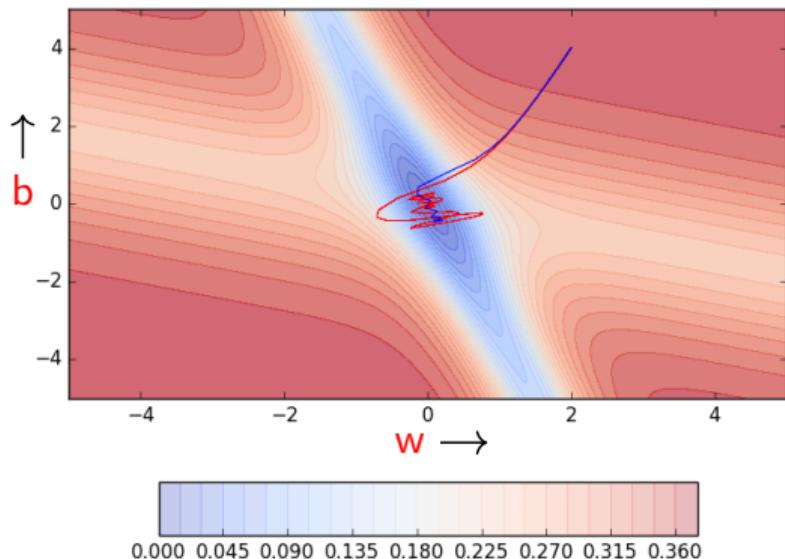
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



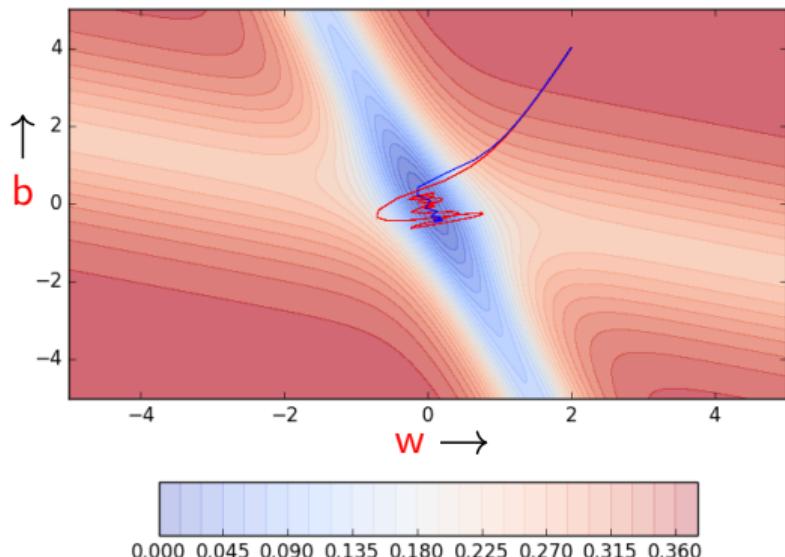
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

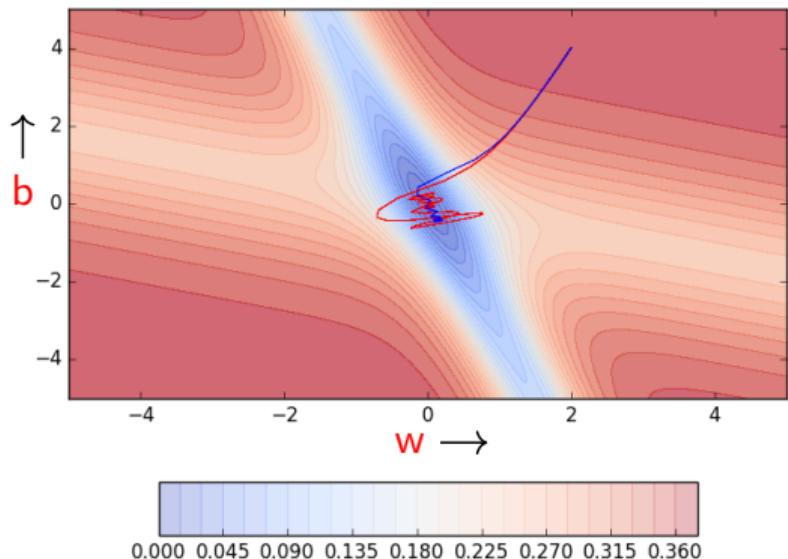


```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

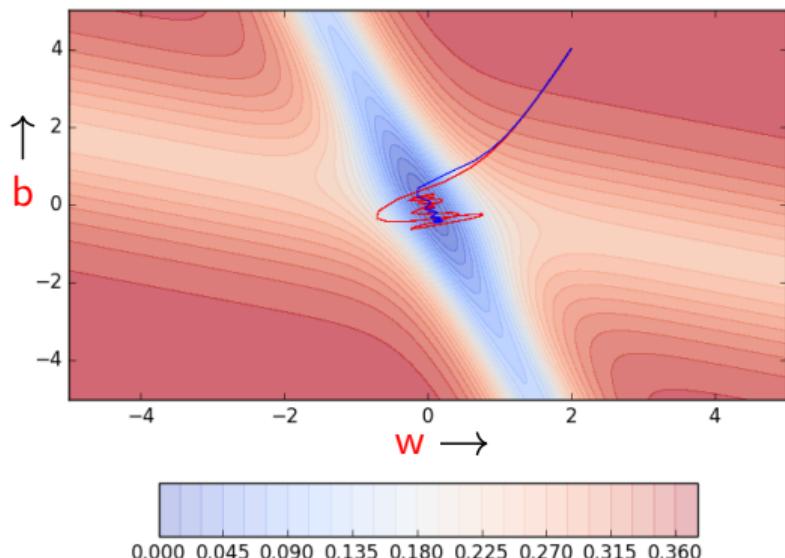




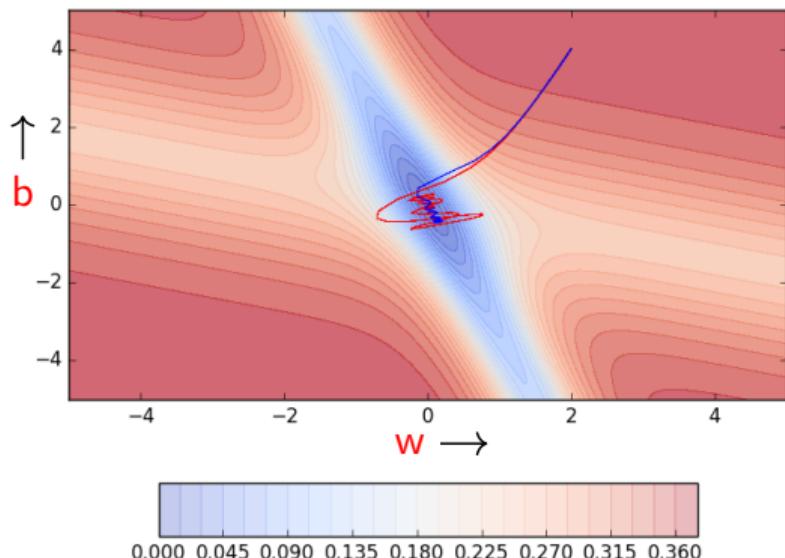
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



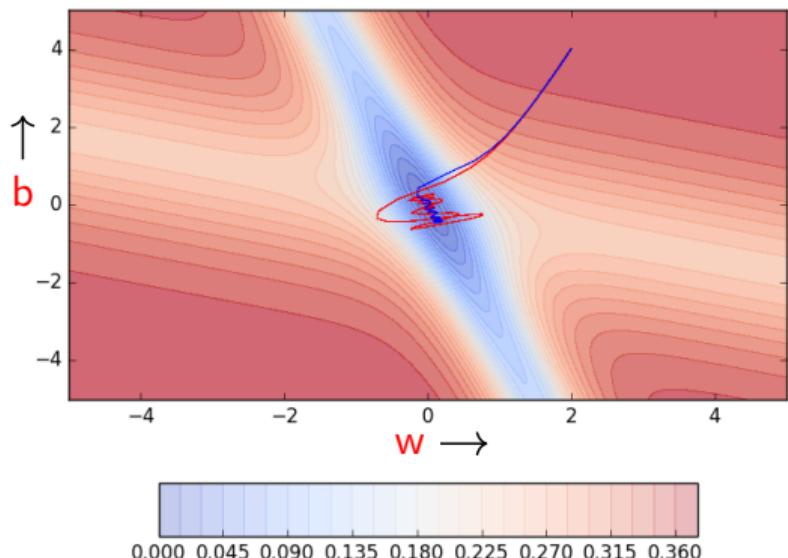
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

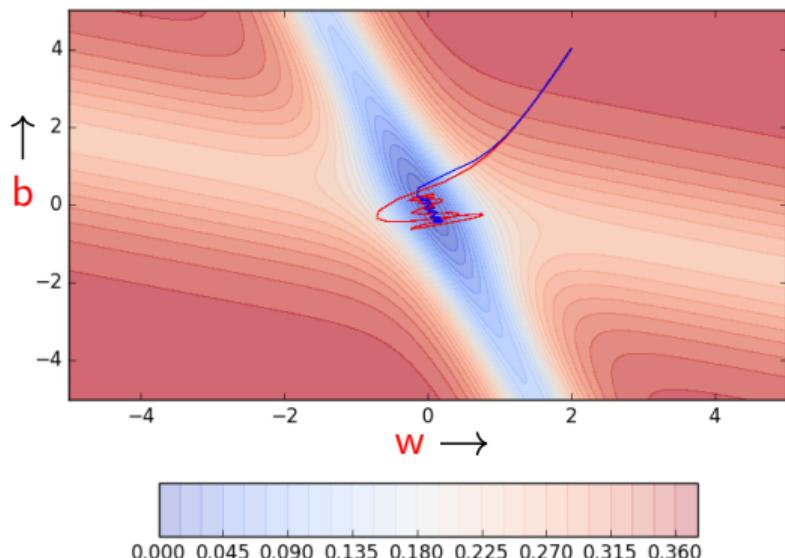


```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

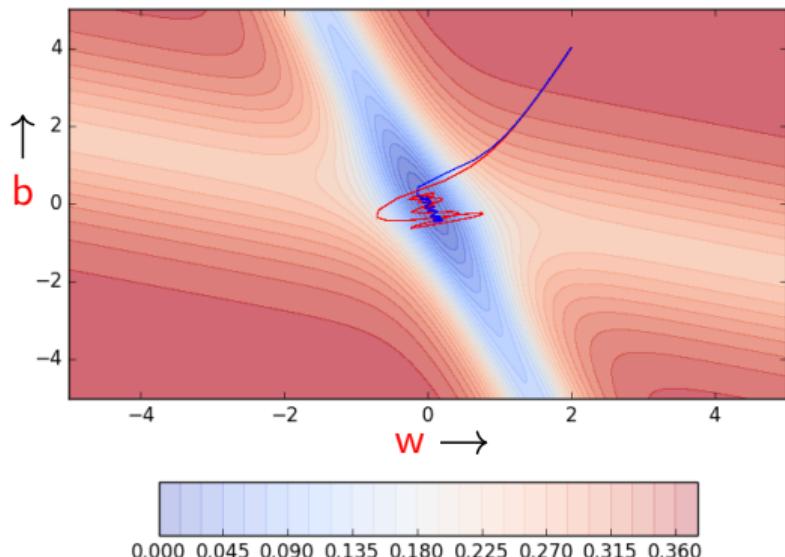




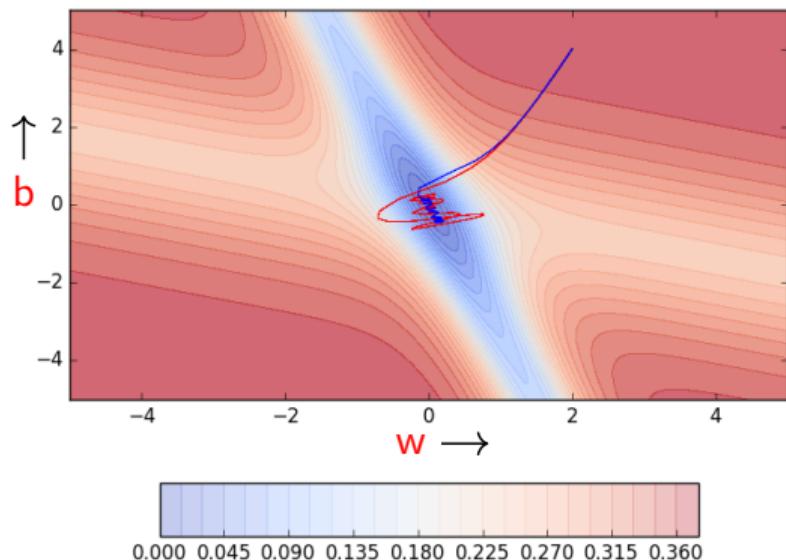
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

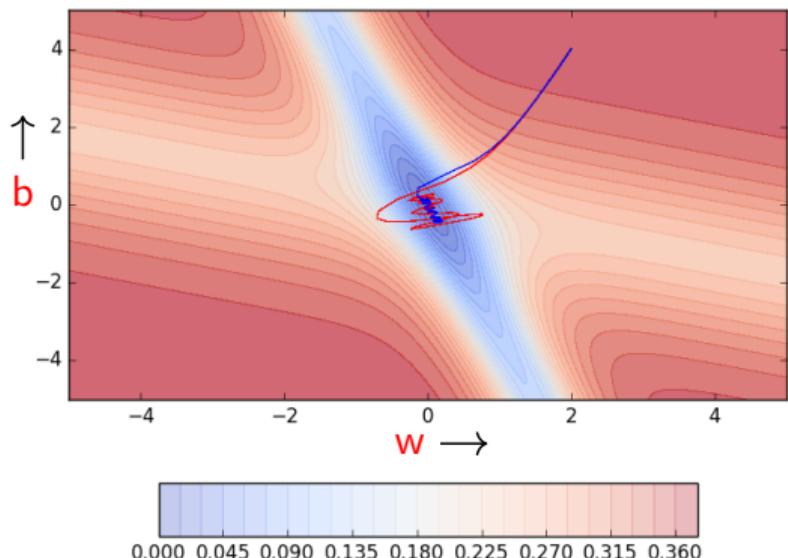


```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



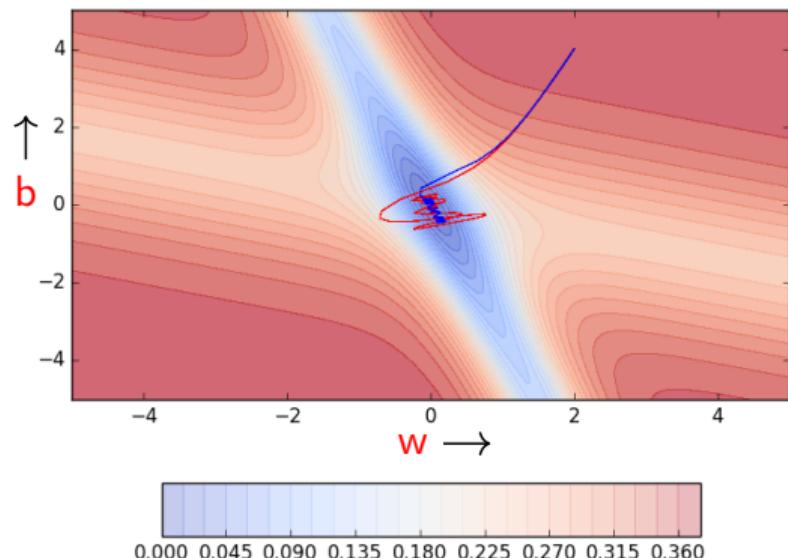


```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

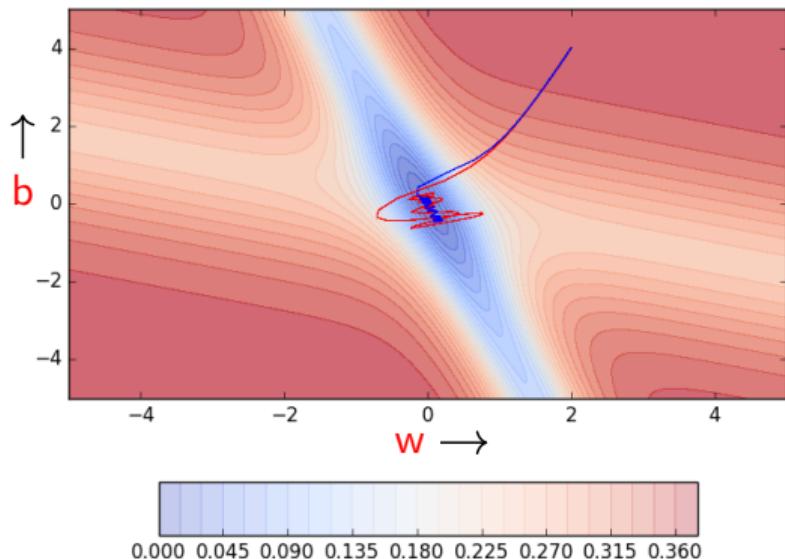




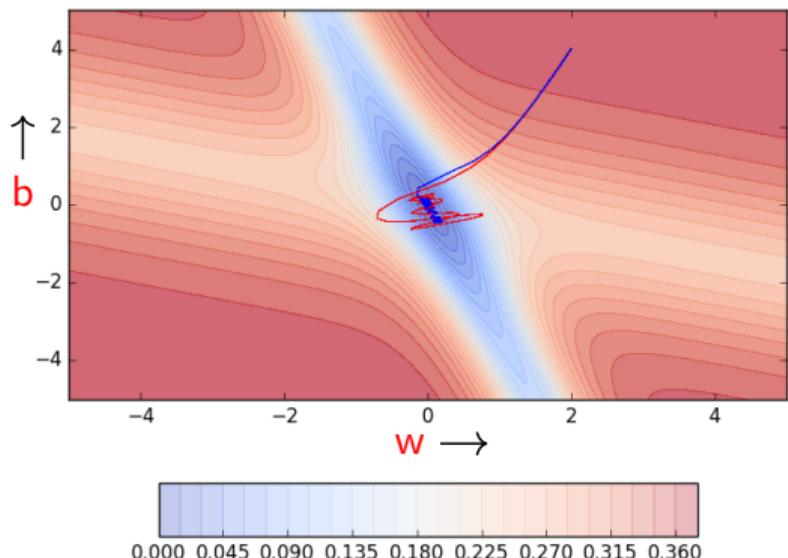
```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

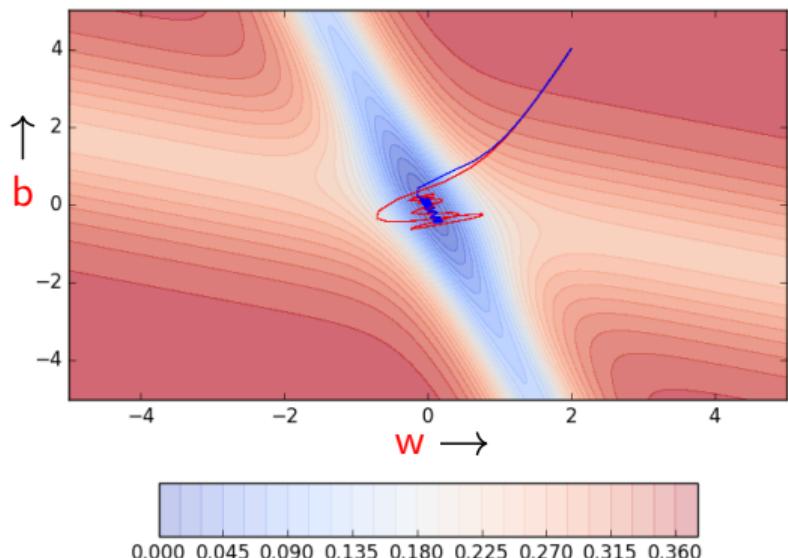


```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

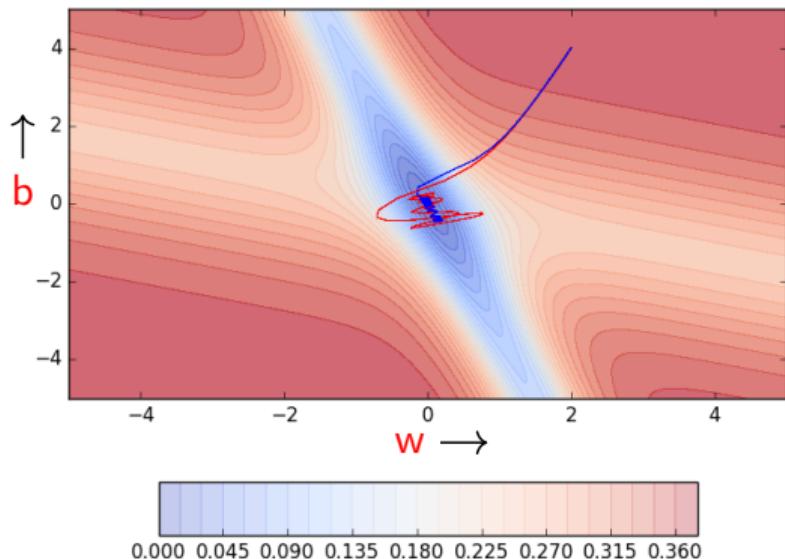




```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```



```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```





随机梯度下降和小批量梯度下降 (Stochastic And Mini-Batch Gradient Descent)

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db
```

```
X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db
```

- 批量梯度下降 (Batch Gradient Descent) 在所有数据上计算损失函数对于参数的梯度



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db

```



- 批量梯度下降 (Batch Gradient Descent) 在所有数据上计算损失函数对于参数的梯度
- 很慢



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w, b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db

```

- 批量梯度下降 (Batch Gradient Descent) 在所有数据上计算损失函数对于参数的梯度
- 很慢 假设有一百万个数据，对参数的每一次更新，计算梯度时都需要遍历所有所有一千万个训练数据。





```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w,b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db

```

- 批量梯度下降 (Batch Gradient Descent) 在所有数据上计算损失函数对于参数的梯度
- 很慢 假设有一百万个数据，对参数的每一次更新，计算梯度时都需要遍历所有所有一千万个训练数据。
- 当内存不足以装载所有数据时，没法计算梯度，也不适合数据在线得到的场合。





```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w, b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db

```

- 批量梯度下降 (Batch Gradient Descent) 在所有数据上计算损失函数对于参数的梯度
- 很慢 假设有一百万个数据，对参数的每一次更新，计算梯度时都需要遍历所有所有一千万个训练数据。
- 理论保证



```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w, b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db

```



- 批量梯度下降 (Batch Gradient Descent) 在所有数据上计算损失函数对于参数的梯度
- 很慢 假设有一百万个数据，对参数的每一次更新，计算梯度时都需要遍历所有所有一千万个训练数据。
- 理论保证 每一步更新，损失函数都会下降。当损失函数是凸函数时，能够保证收敛到全局的最小值点。对于凸函数，能够收敛到局部最小值点

```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w, b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db

```



- 批量梯度下降 (Batch Gradient Descent) 在所有数据上计算损失函数对于参数的梯度
- 很慢 假设有一百万个数据，对参数的每一次更新，计算梯度时都需要遍历所有所有一千万个训练数据。
- 理论保证 每一步更新，损失函数都会下降。当损失函数是凸函数时，能够保证收敛到全局的最小值点。对于凸函数，能够收敛到局部最小值点
- 如何克服速度慢的问题？

```

X = [0.5, 2.5]
Y = [0.2, 0.9]

def f(w, b, x): #sigmoid with parameters w, b
    return 1.0 / (1.0 + np.exp(-(w*x +b)))

def error(w, b):
    err = 0.0
    for x,y in zip(X,Y):
        fx = f(w,b,x)
        err += 0.5* (fx - y) ** 2
    return err

def grad_b(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx)

def grad_w(w, b, x, y):
    fx = f(w, b, x)
    return (fx - y) * fx * (1 - fx) * x

def do_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db

```



- 批量梯度下降 (Batch Gradient Descent) 在所有数据上计算损失函数对于参数的梯度
- 很慢 假设有一百万个数据，对参数的每一次更新，计算梯度时都需要遍历所有所有一千万个训练数据。
- 理论保证 每一步更新，损失函数都会下降。当损失函数是凸函数时，能够保证收敛到全局的最小值点。对于凸函数，能够收敛到局部最小值点
- 如何克服速度慢的问题？ 随机梯度下降 (stochastic gradient descent)

```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
            w = w - eta * dw
            b = b - eta * db
```

```
def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db
```

```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
            w = w - eta * dw
            b = b - eta * db
```

- 使用每一个数据，对参数进行更新

```
def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db
```



```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
            w = w - eta * dw
            b = b - eta * db
```

- 使用每一个数据，对参数进行更新
- 如果有一百万个训练数据，每下个 epoch，需要更新参数一百万次 (1 epoch = 1 pass over the data; 1 step = 1 update)

```
def do_gradient_descent() :
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
        w = w - eta * dw
        b = b - eta * db
```

```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
            w = w - eta * dw
            b = b - eta * db
```



- 使用每一个数据，对参数进行更新
- 如果有一百万个训练数据，每下个 epoch，需要更新参数一百万次 (1 epoch = 1 pass over the data; 1 step = 1 update)
- 缺点？

```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
            w = w - eta * dw
            b = b - eta * db
```



- 使用每一个数据，对参数进行更新
- 如果有一百万个训练数据，每下个 epoch，需要更新参数一百万次 (1 epoch = 1 pass over the data; 1 step = 1 update)
- 缺点？近似梯度

```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
            w = w - eta * dw
            b = b - eta * db
```



- 『随机』的意思是说，使用一个训练数据来估计整个梯度

- 使用每一个数据，对参数进行更新
- 如果有一百万个训练数据，每下个 epoch，需要更新参数一百万次 (1 epoch = 1 pass over the data; 1 step = 1 update)
- 缺点？近似梯度



```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
            w = w - eta * dw
            b = b - eta * db
```

- 『随机』的意思是说，使用一个训练数据来估计整个梯度
类似于，抛一次硬币来估计 $P(\text{heads})$.

- 使用每一个数据，对参数进行更新
- 如果有一百万个训练数据，每下个 epoch，需要更新参数一百万次 (1 epoch = 1 pass over the data; 1 step = 1 update)
- 缺点？近似梯度



```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
            w = w - eta * dw
            b = b - eta * db
```

- 『随机』的意思是说，使用一个训练数据来估计整个梯度
类似于，抛一次硬币来估计 $P(\text{heads})$.

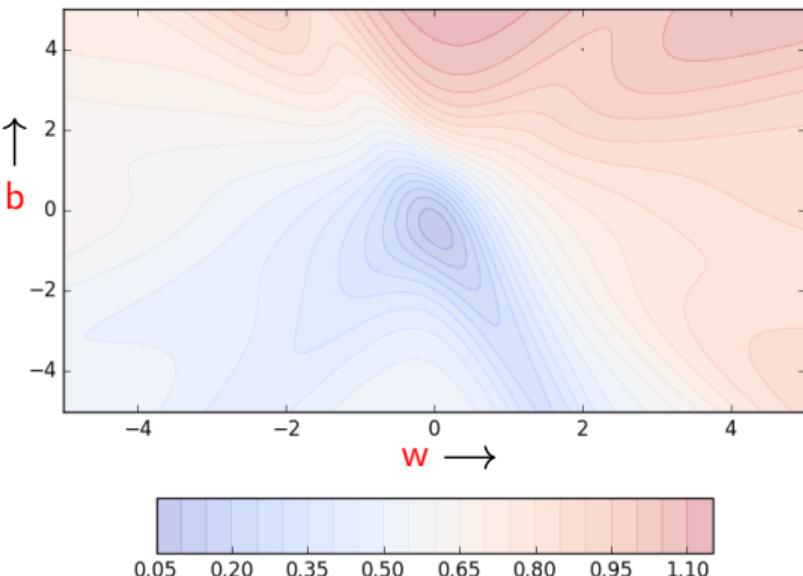
- 使用每一个数据，对参数进行更新
- 如果有一百万个训练数据，每下个 epoch，需要更新参数一百万次 (1 epoch = 1 pass over the data; 1 step = 1 update)
- 缺点？近似梯度
- 不能保证每一次更新会降低损失函数

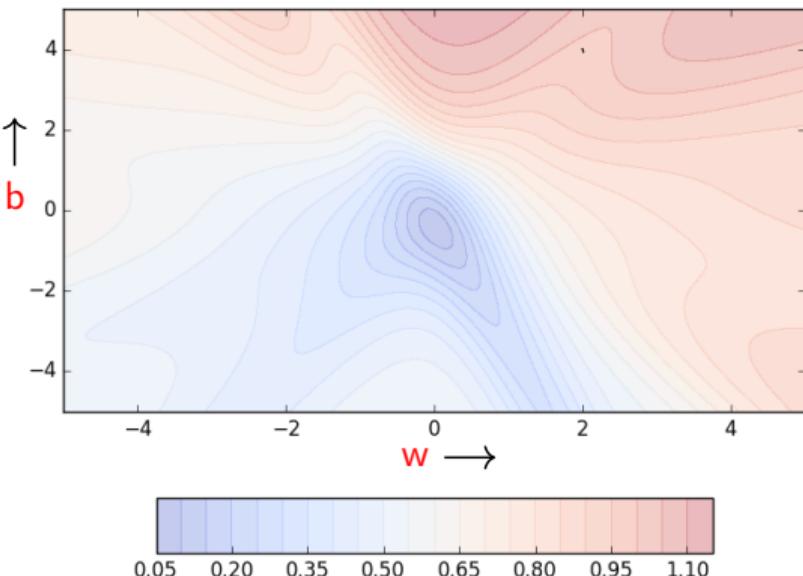
```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

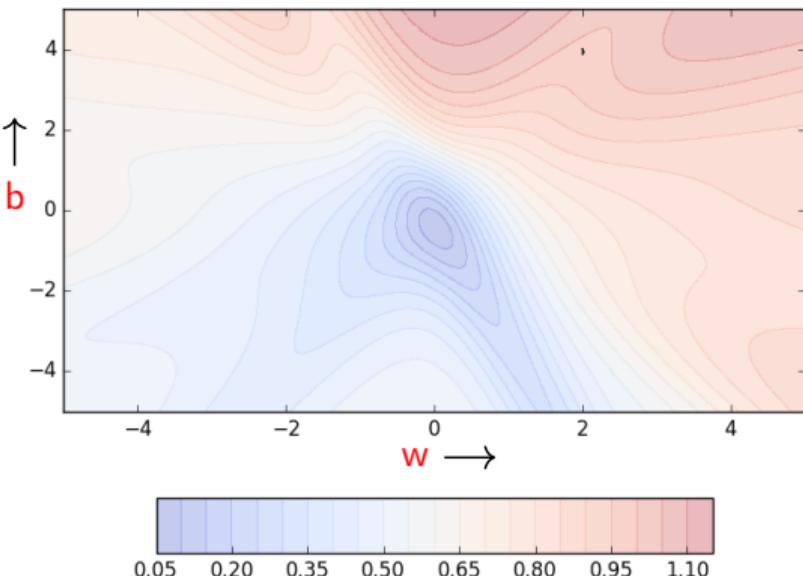


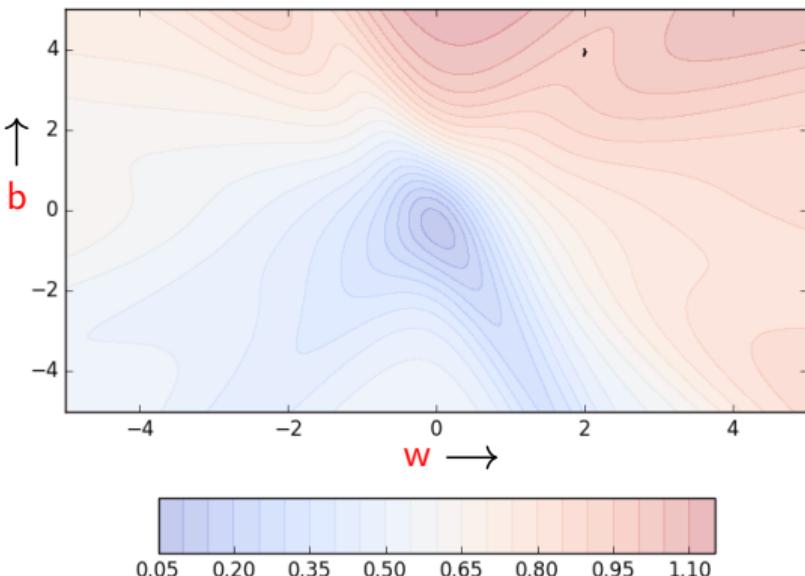
- 『随机』的意思是说，使用一个训练数据来估计整个梯度类似于，抛一次硬币来估计 $P(\text{heads})$.

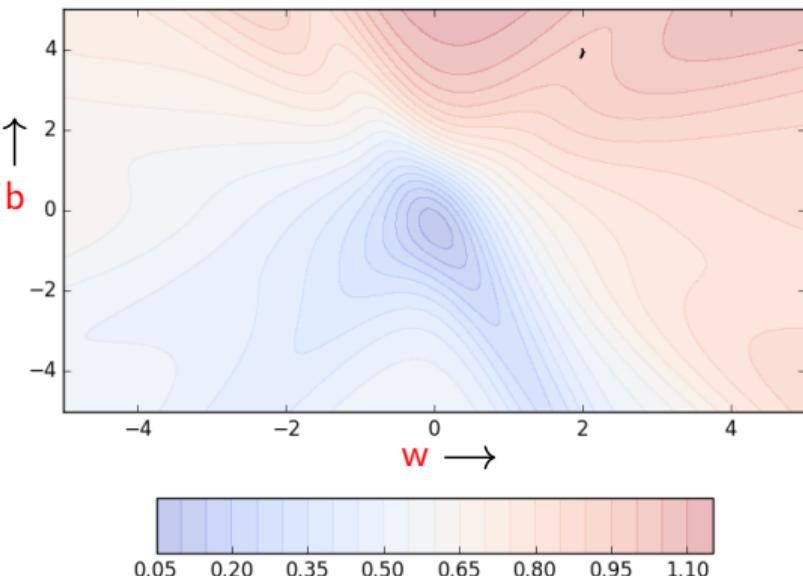
- 使用每一个数据，对参数进行更新
- 如果有一百万个训练数据，每下个 epoch，需要更新参数一百万次 (1 epoch = 1 pass over the data; 1 step = 1 update)
- 缺点？近似梯度
- 不能保证每一次更新会降低损失函数
- 看一个例子

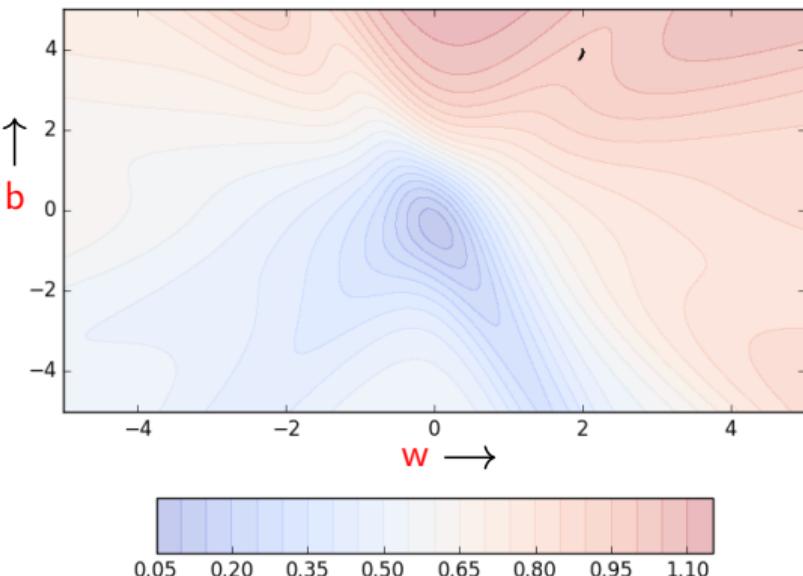


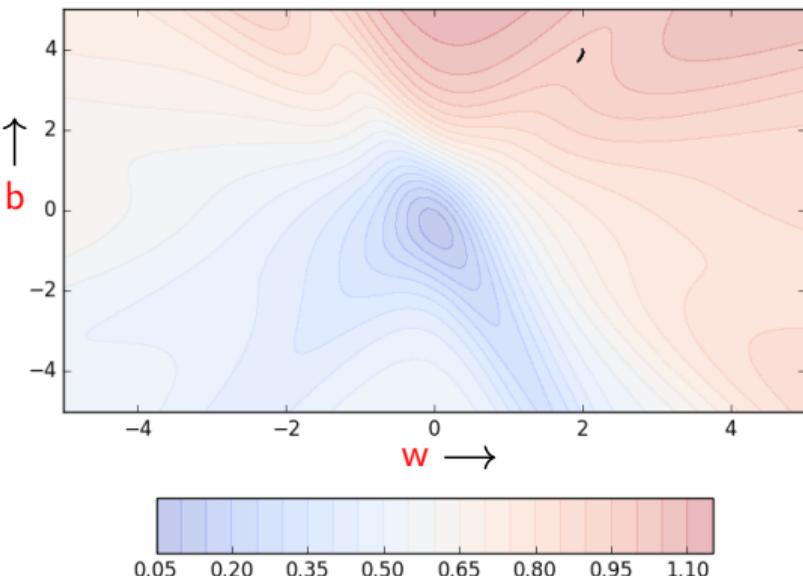


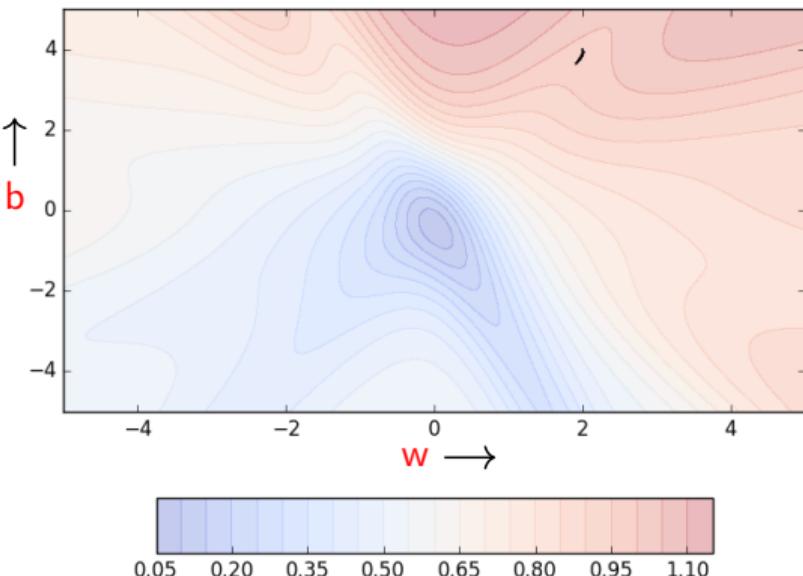


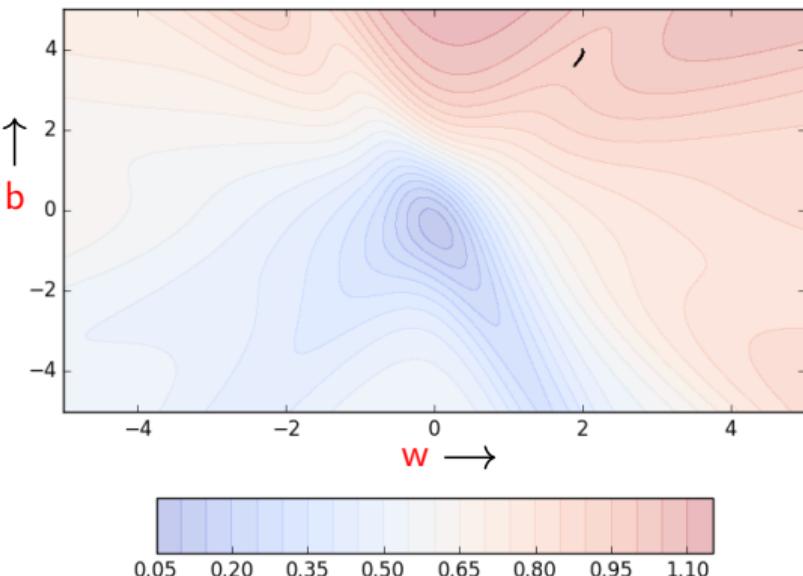


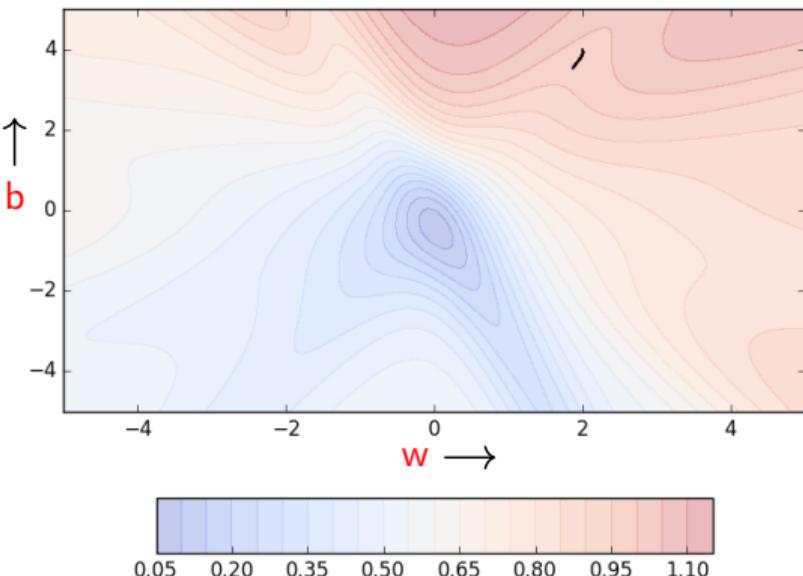


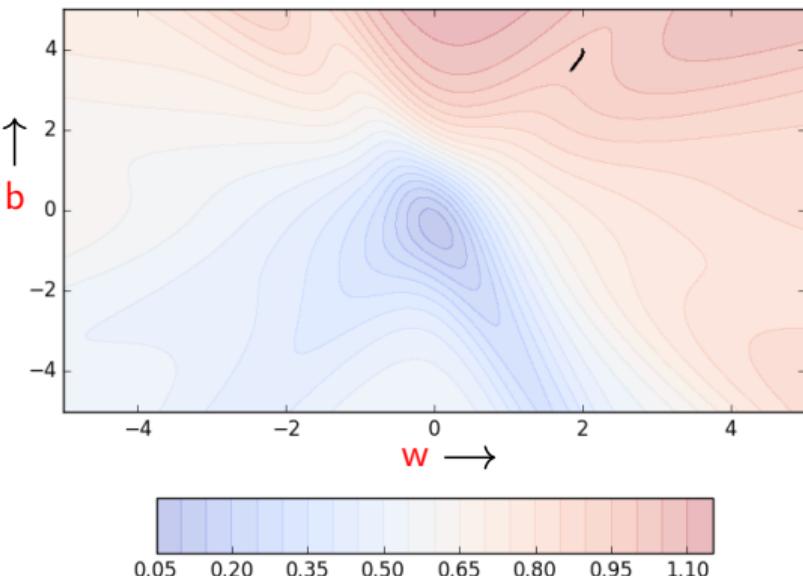


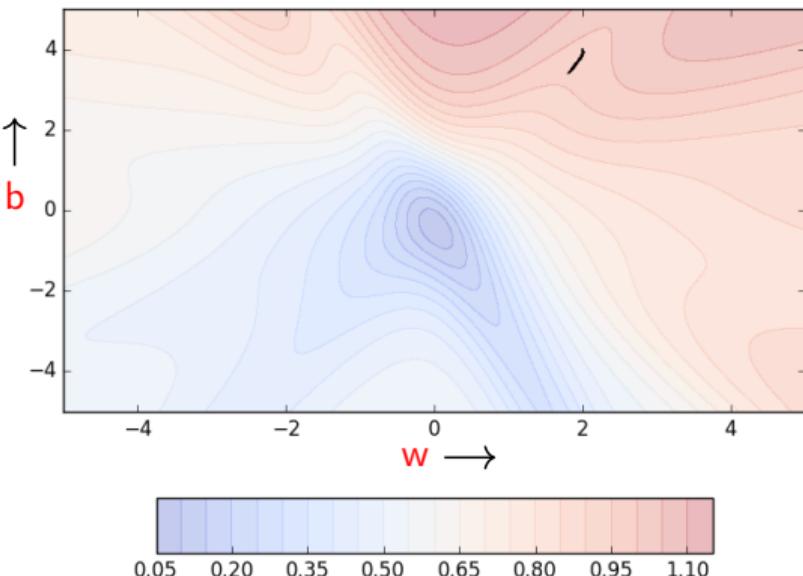


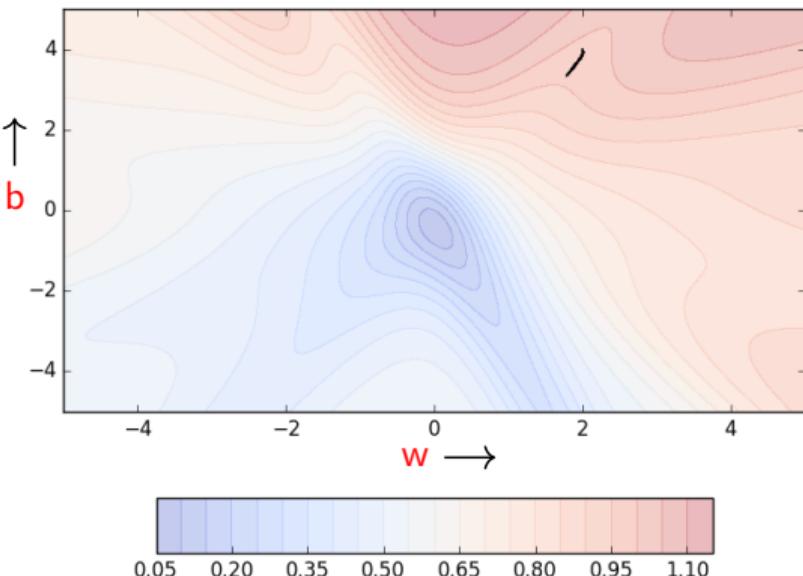


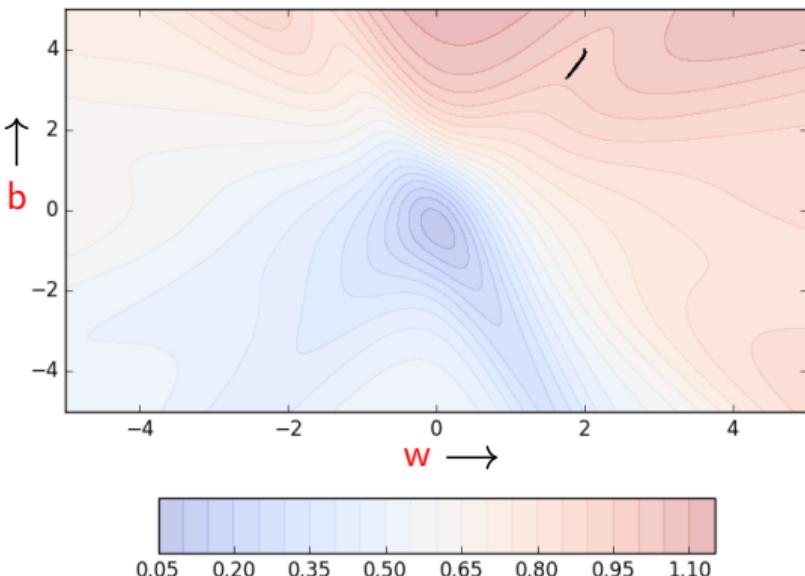


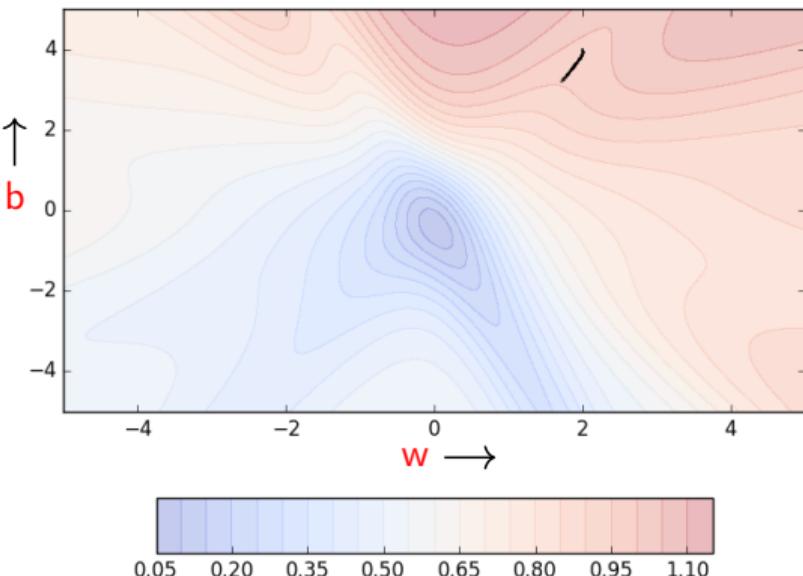


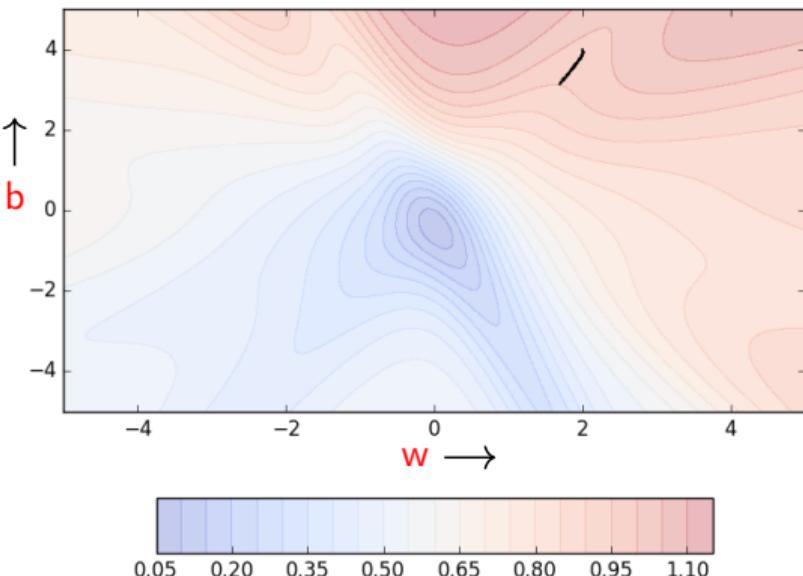


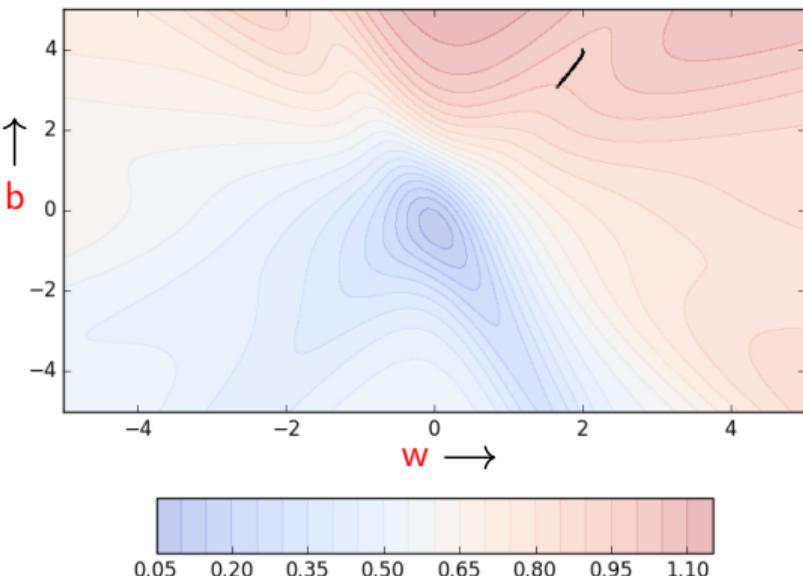


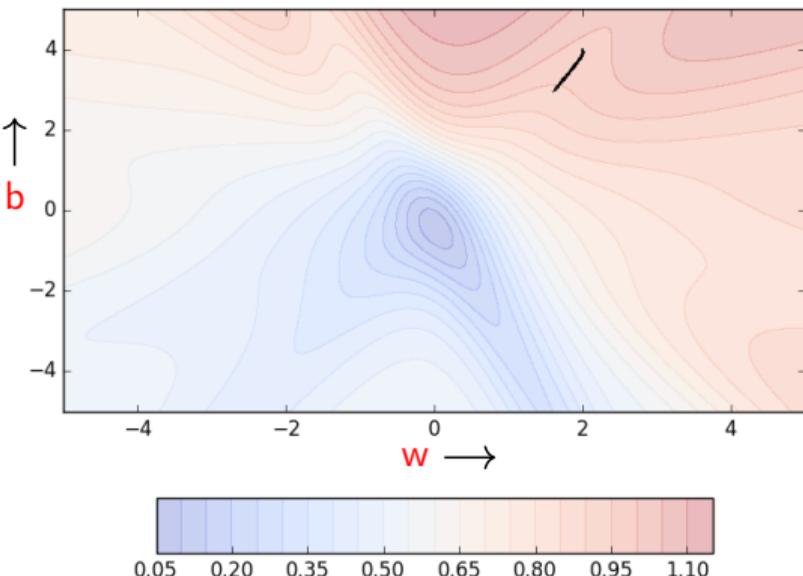


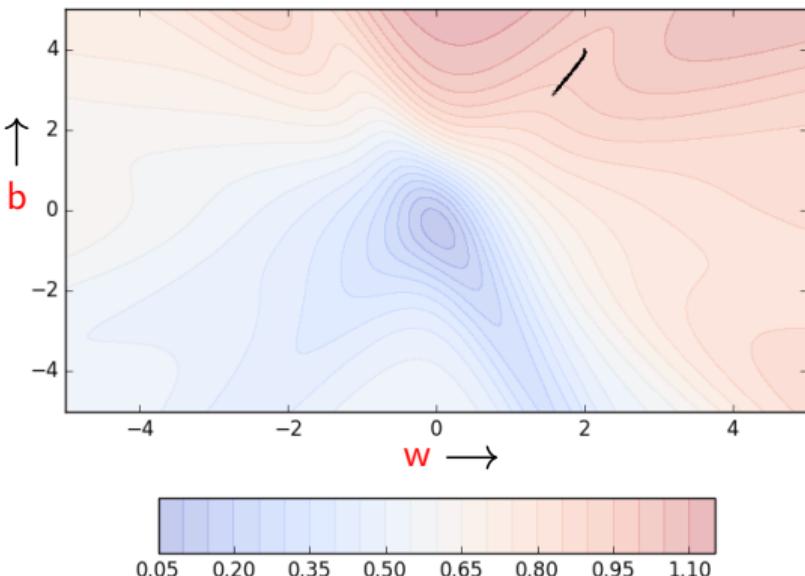


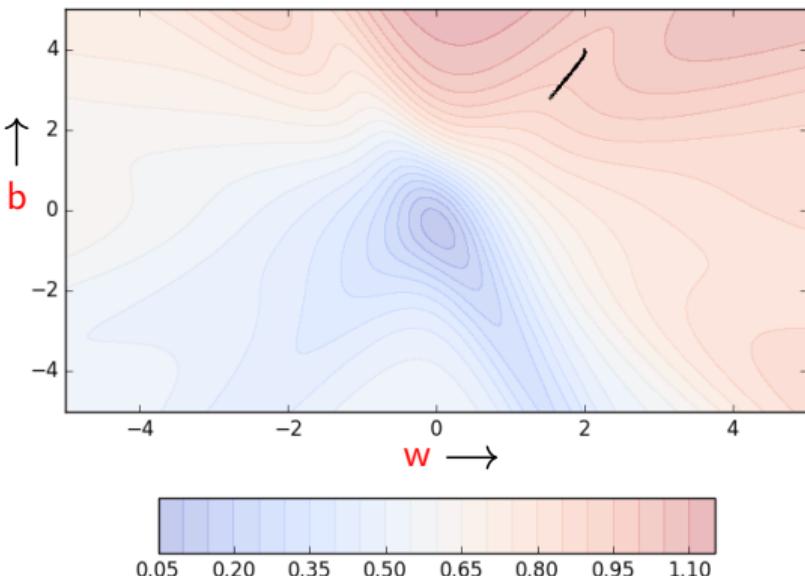


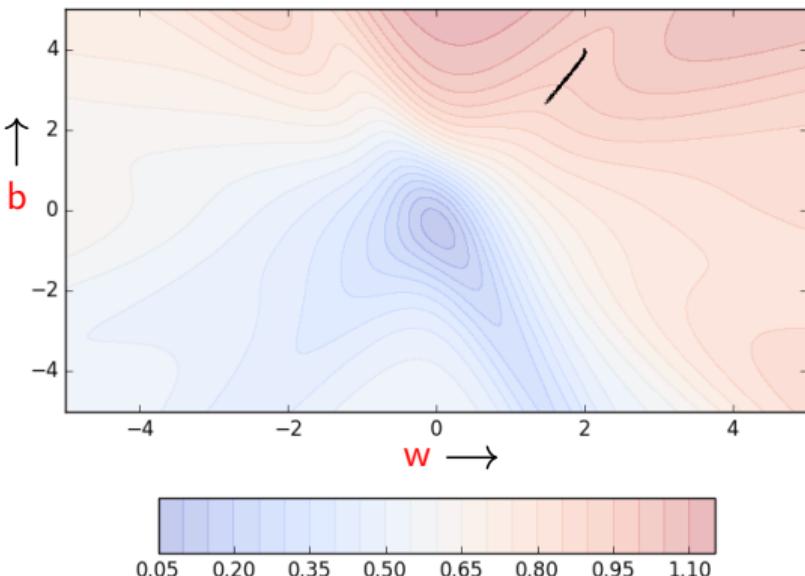


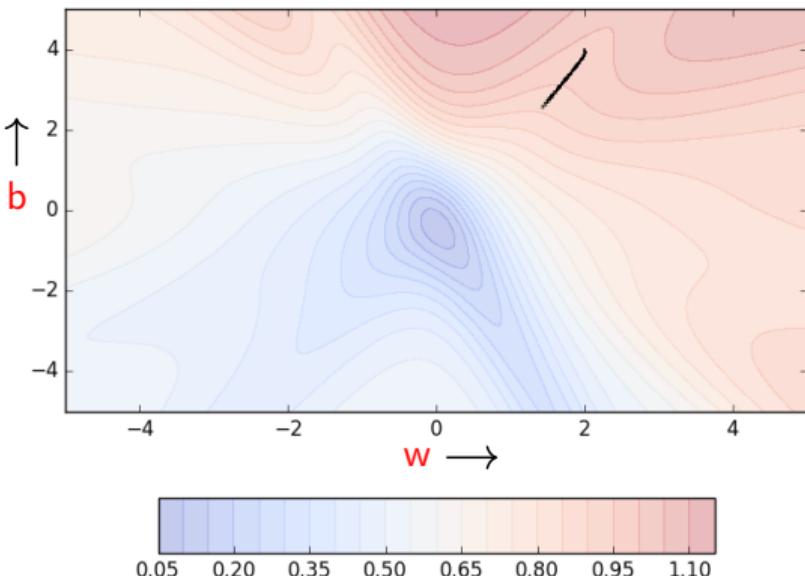


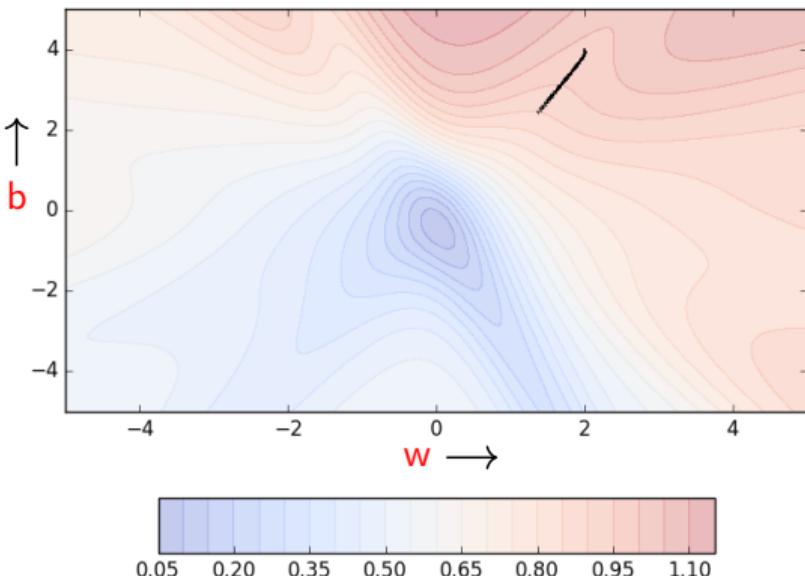


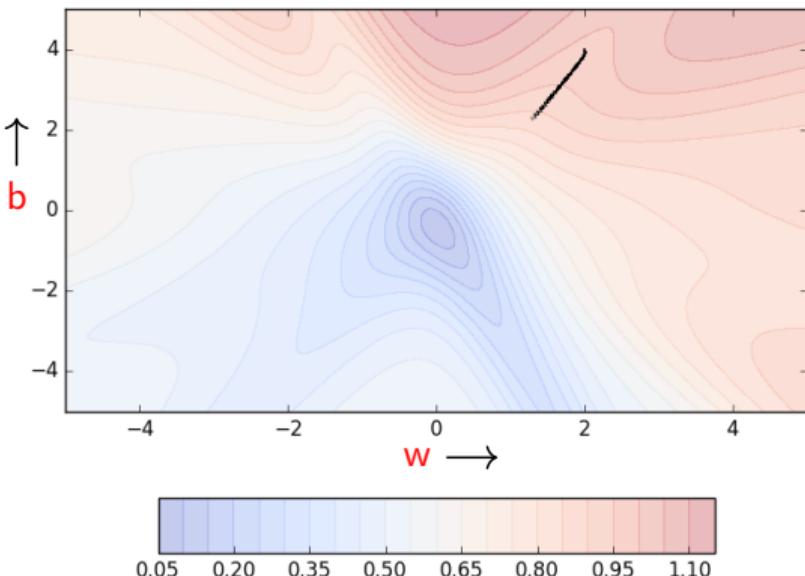


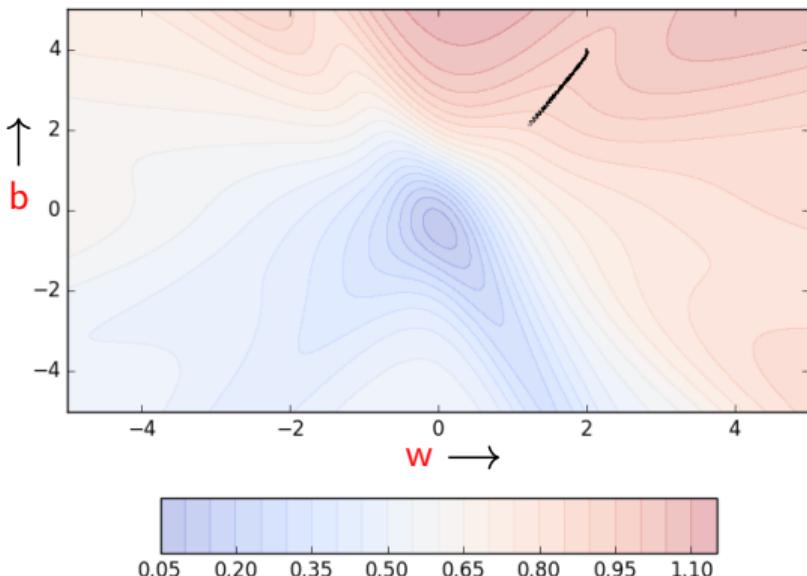


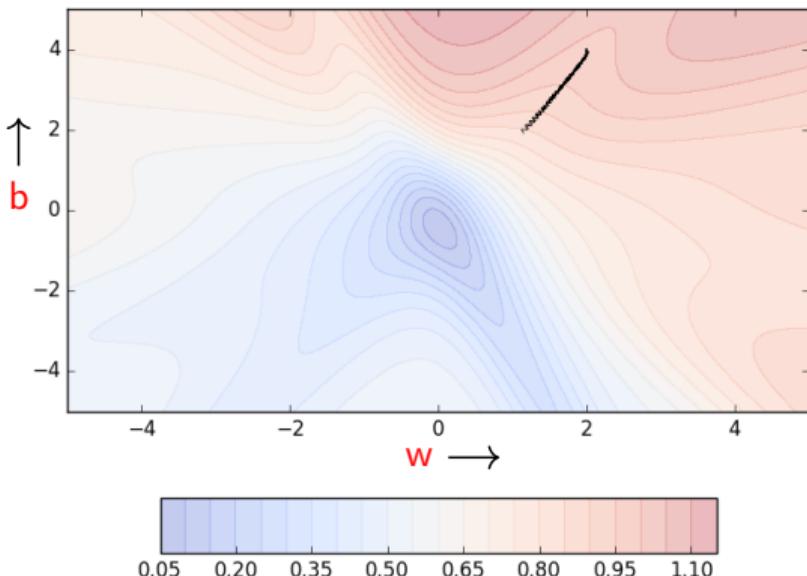


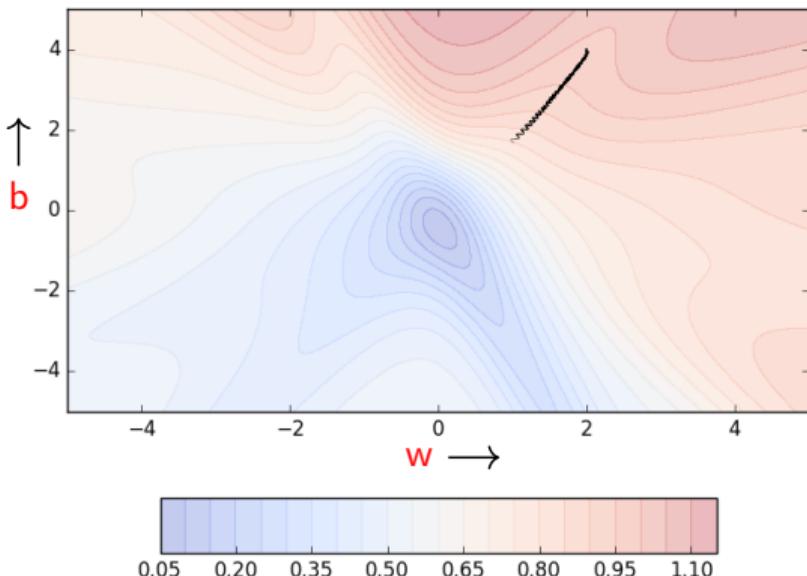


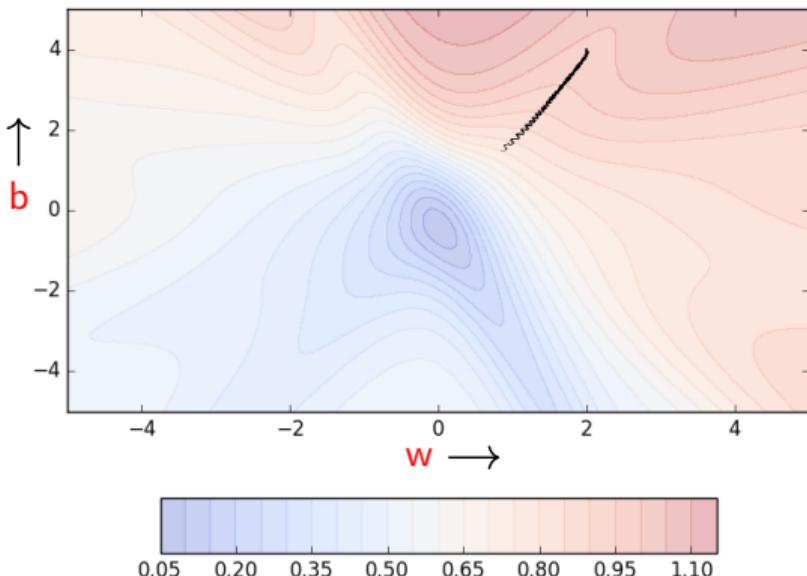


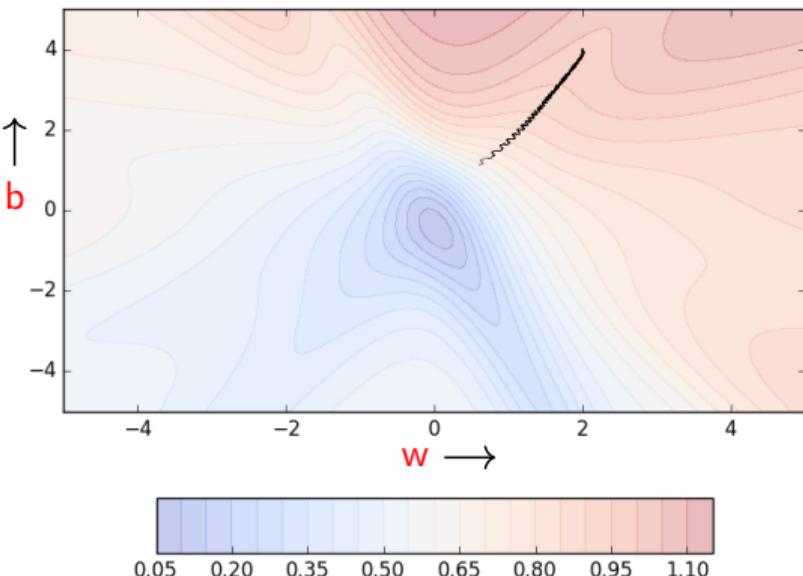


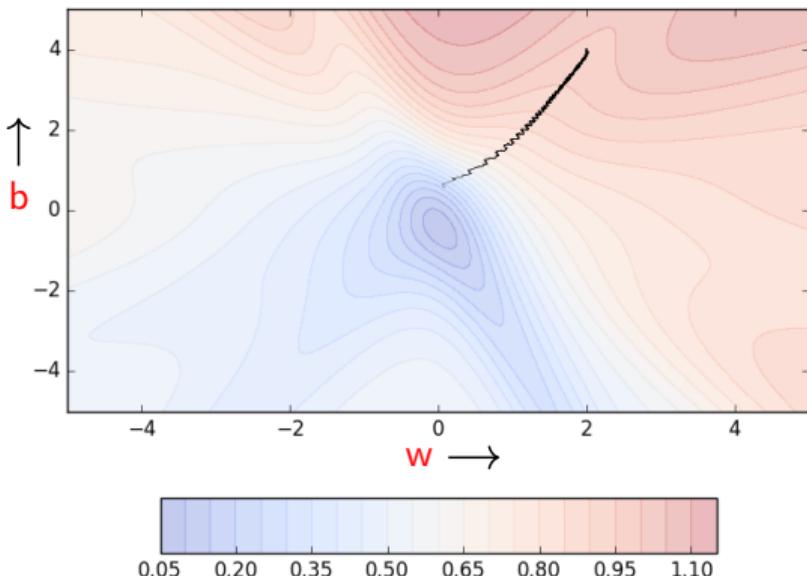


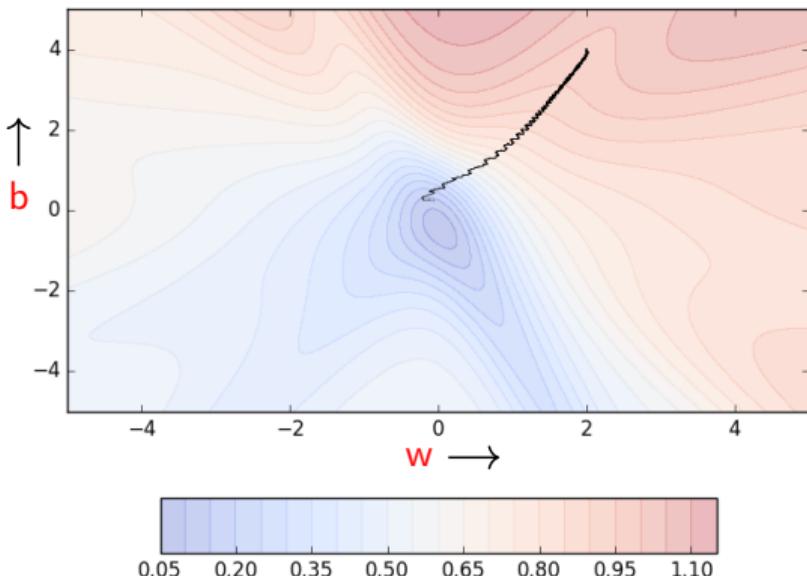


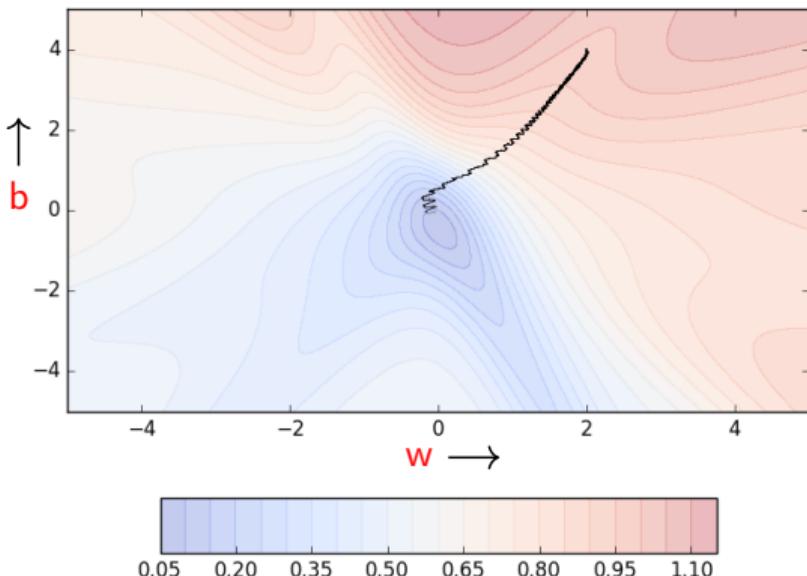


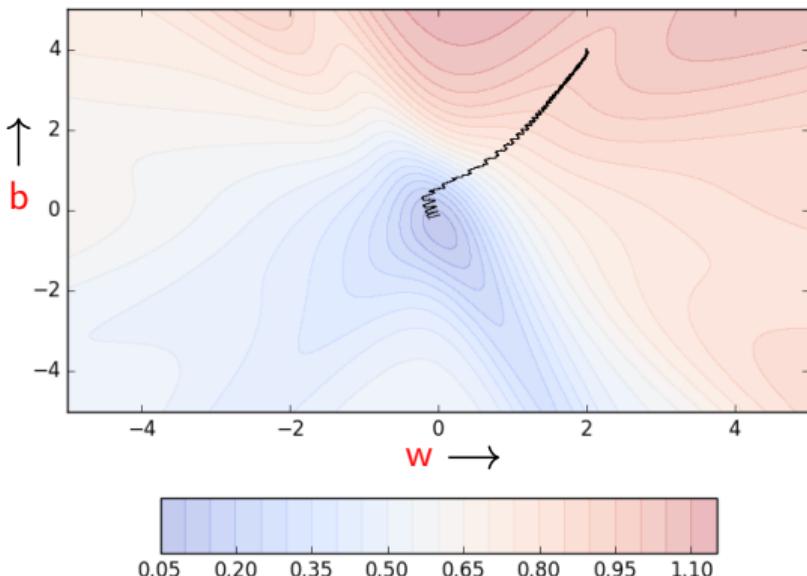


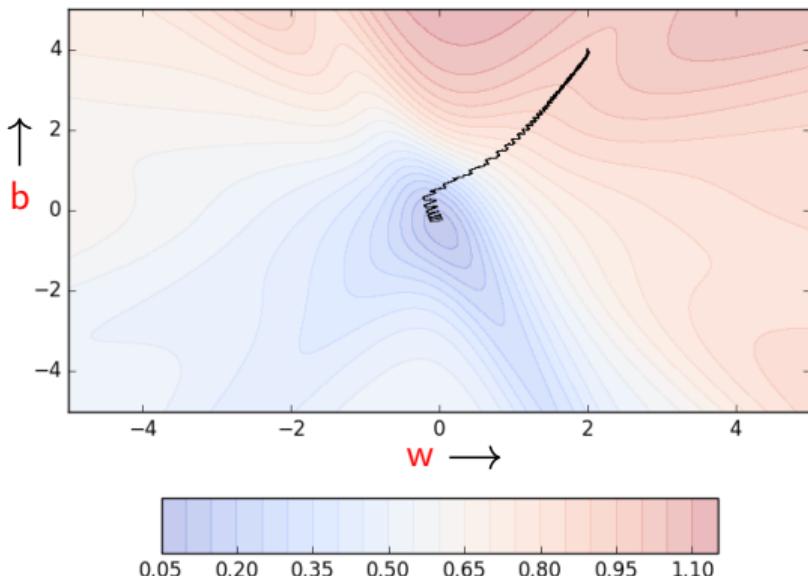


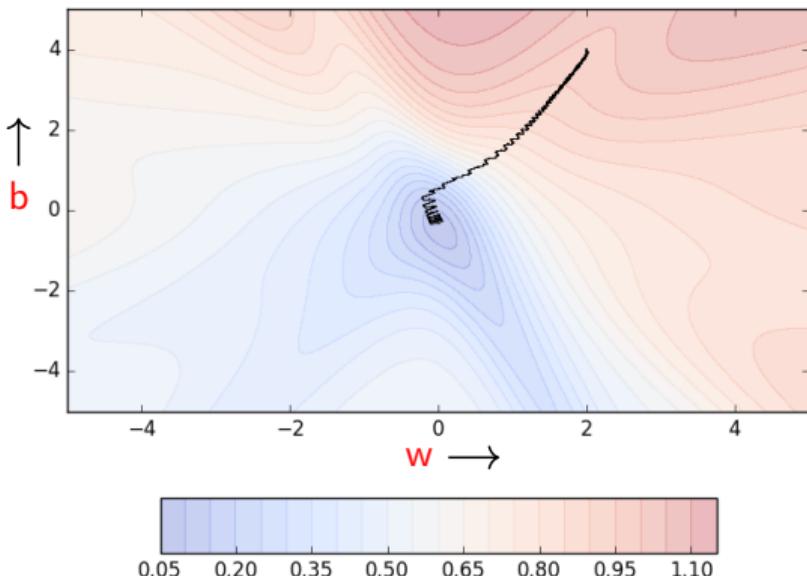


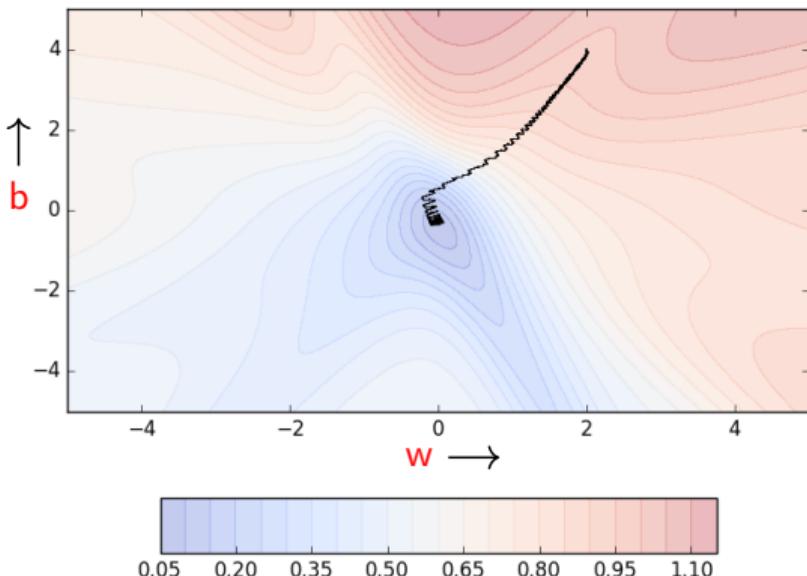




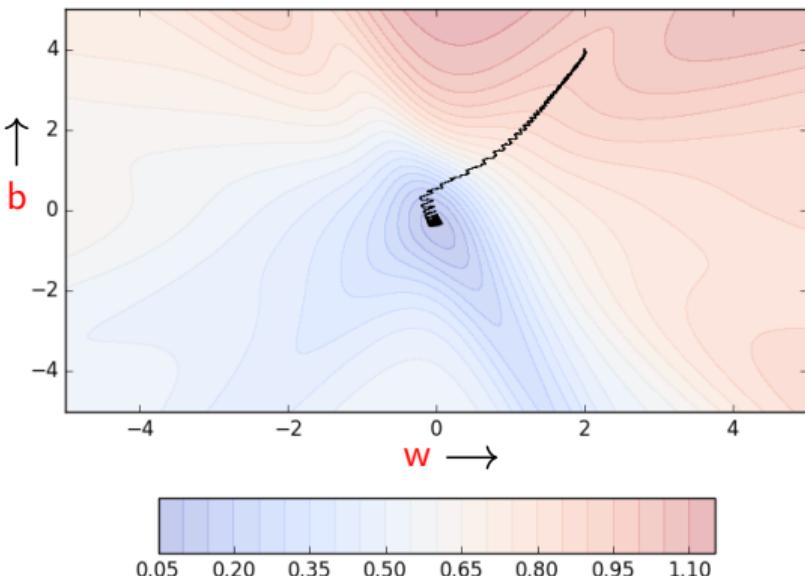






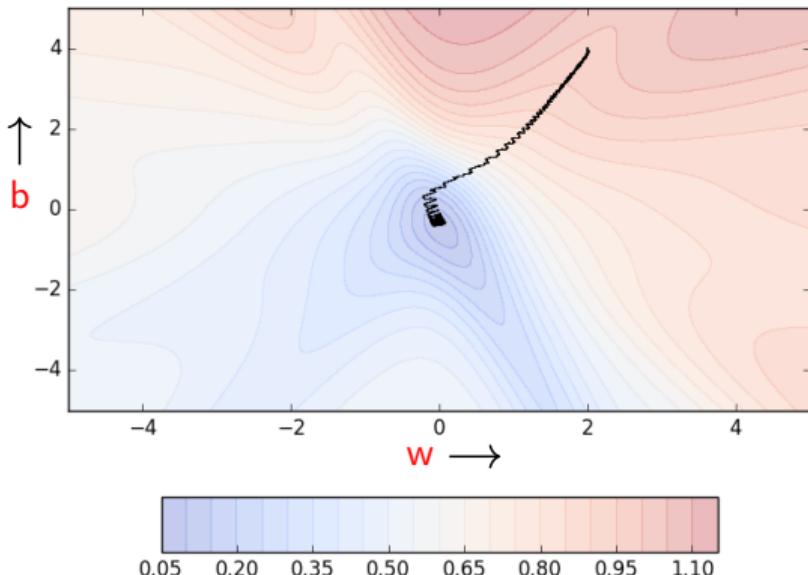


- 我们能看到很多振荡。为什么？



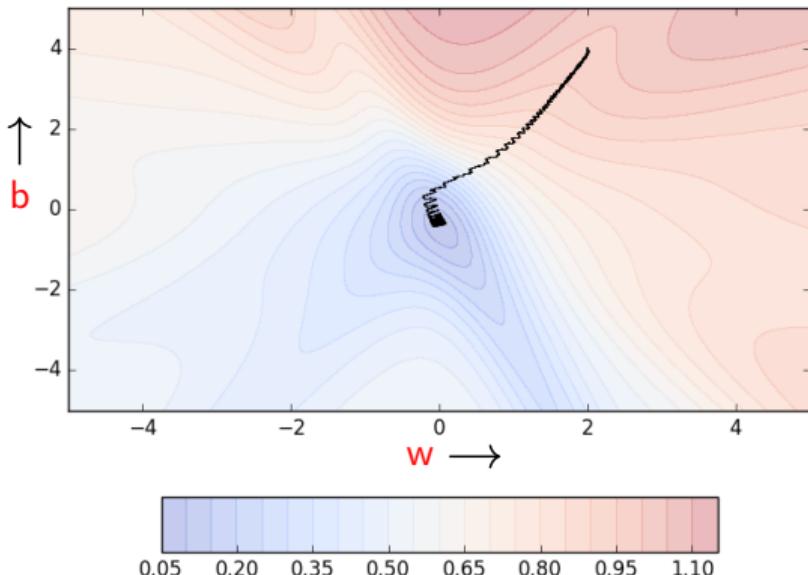


- 我们能看到很多振荡。为什么？
- 每一个训练数据都在朝着它喜欢的方向对参数进行更新



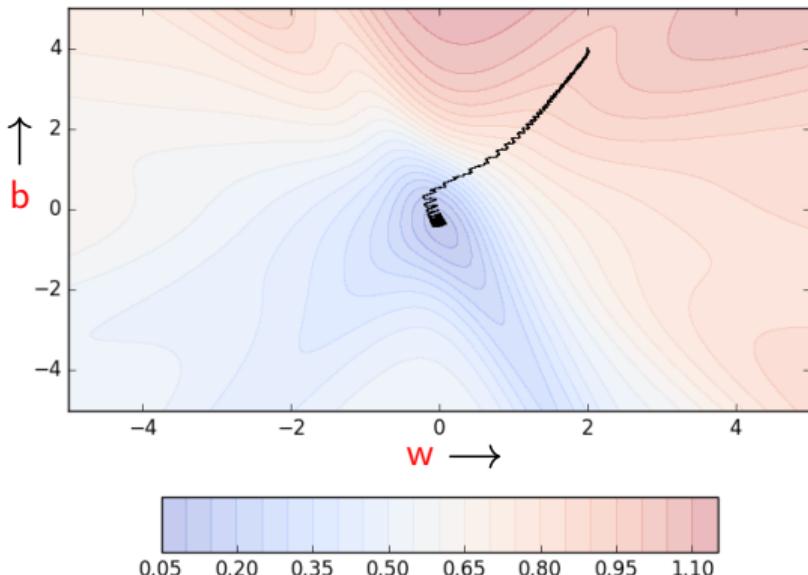


- 我们能看到很多振荡。为什么？
- 每一个训练数据都在朝着它喜欢的方向对参数进行更新（而没有考虑到对其他数据的影响）

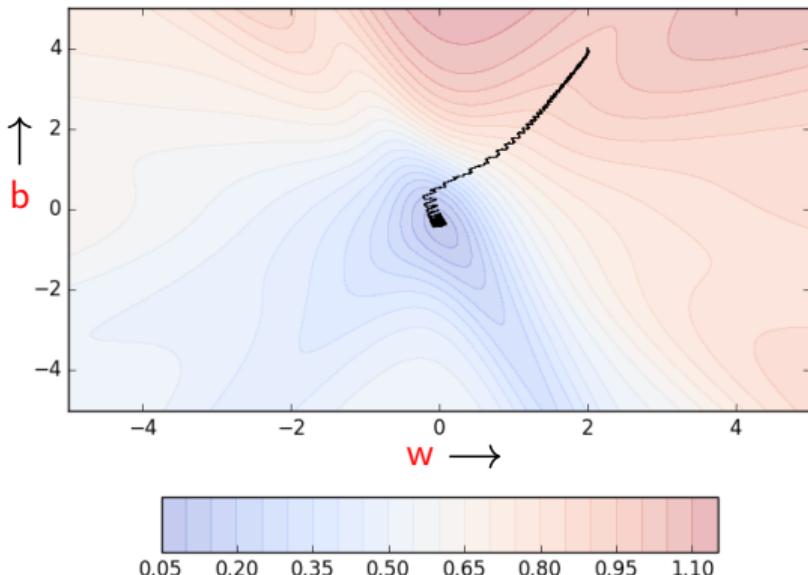




- 我们能看到很多振荡。为什么？
- 每一个训练数据都在朝着它喜欢的方向对参数进行更新（而没有考虑到对其他数据的影响）
- 好像所有训练数据在互相竞争

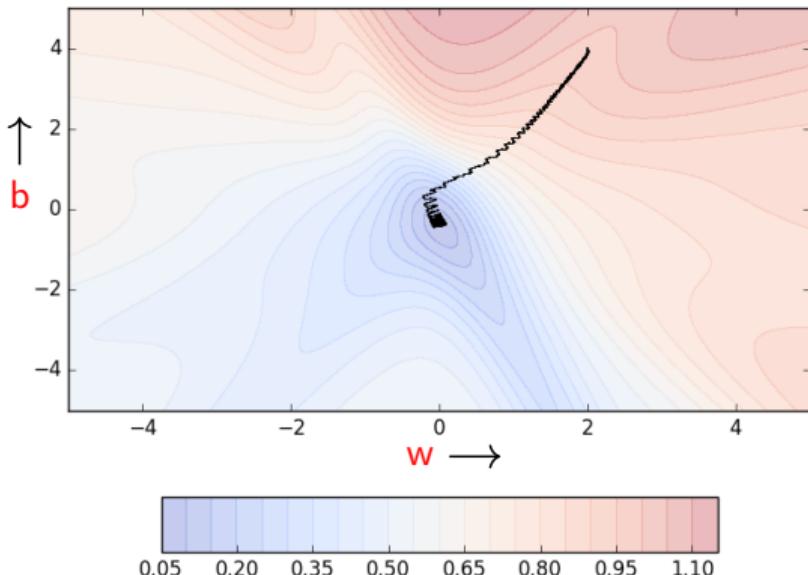


- 我们能看到很多振荡。为什么？
- 每一个训练数据都在朝着它喜欢的方向对参数进行更新（而没有考虑到对其他数据的影响）
- 好像所有训练数据在互相竞争
- 能否通过改进梯度的计算来减少振荡？



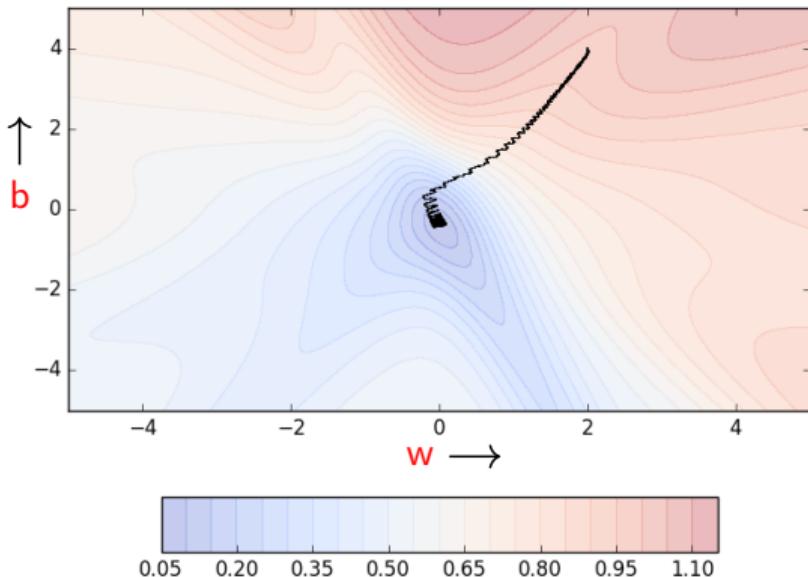


- 我们能看到很多振荡。为什么？
- 每一个训练数据都在朝着它喜欢的方向对参数进行更新（而没有考虑到对其他数据的影响）
- 好像所有训练数据在互相竞争
- 能否通过改进梯度的计算来减少振荡？
(现在每次使用一个数据来计算梯度)

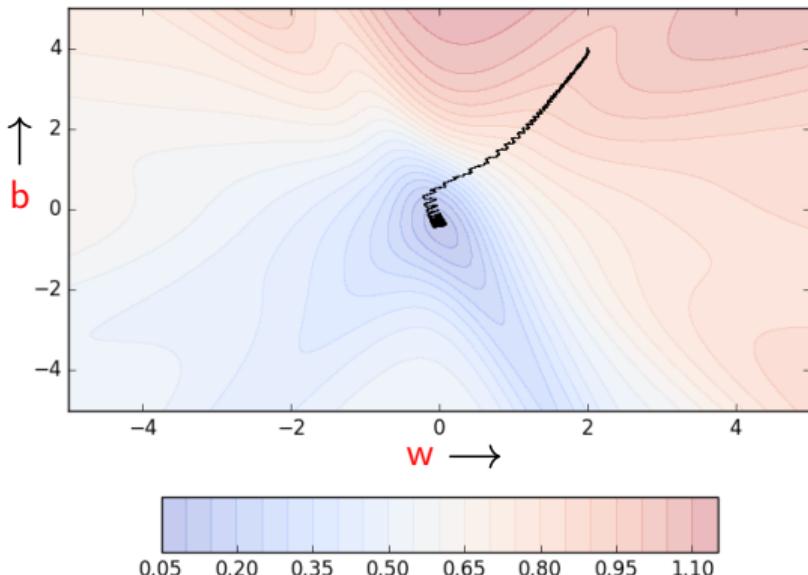




- 我们能看到很多振荡。为什么？
- 每一个训练数据都在朝着它喜欢的方向对参数进行更新（而没有考虑到对其他数据的影响）
- 好像所有训练数据在互相竞争
- 能否通过改进梯度的计算来减少振荡？（现在每次使用一个数据来计算梯度）
- 小批量梯度下降（mini-batch gradient descent）

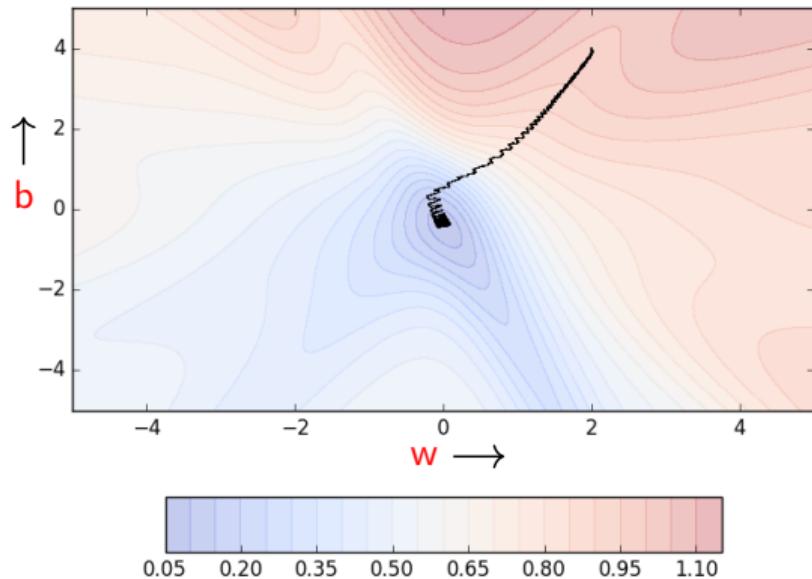


- 我们能看到很多振荡。为什么？
- 每一个训练数据都在朝着它喜欢的方向对参数进行更新（而没有考虑到对其他数据的影响）
- 好像所有训练数据在互相竞争
- 能否通过改进梯度的计算来减少振荡？（现在每次使用一个数据来计算梯度）
- 小批量梯度下降（mini-batch gradient descent）





- 我们能看到很多振荡。为什么？
- 每一个训练数据都在朝着它喜欢的方向对参数进行更新（而没有考虑到对其他数据的影响）
- 好像所有训练数据在互相竞争
- 能否通过改进梯度的计算来减少振荡？（现在每次使用一个数据来计算梯度）
- 小批量梯度下降（mini-batch gradient descent）



```
def do_mini_batch_gradient_descent():
    w, b, eta = -2, -2, 1.0
    mini_batch_size, num_points_seen = 2, 0
    for i in range(max_epochs):
        dw, db, num_points = 0, 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
            num_points += 1

        if num_points_seen % mini_batch_size == 0:
            # seen one mini_batch
            w = w - eta * dw
            b = b - eta * db
            dw, db = 0, 0 #reset gradients
```

- 使用 *mini_batch_size* 个数据，对参数进行更新

```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
            w = w - eta * dw
            b = b - eta * db
```

```
def do_mini_batch_gradient_descent():
    w, b, eta = -2, -2, 1.0
    mini_batch_size, num_points_seen = 2, 0
    for i in range(max_epochs):
        dw, db, num_points = 0, 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
            num_points_seen += 1

            if num_points_seen % mini_batch_size == 0:
                # seen one mini_batch
                w = w - eta * dw
                b = b - eta * db
                dw, db = 0, 0 #reset gradients
```

- 使用 *mini_batch_size* 个数据，对参数进行更新
- 比随机梯度下降只使用一个数据对参数更新会更好一些

```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    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)
            w = w - eta * dw
            b = b - eta * db
```

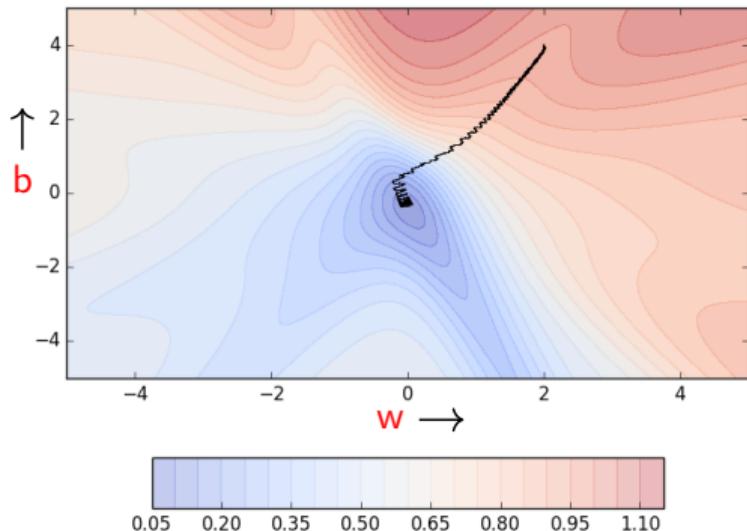


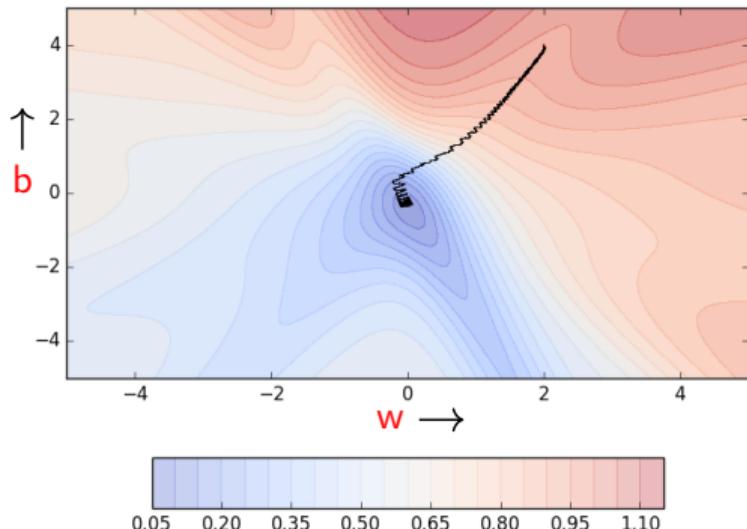
```
def do_mini_batch_gradient_descent():
    w, b, eta = -2, -2, 1.0
    mini_batch_size, num_points_seen = 2, 0
    for i in range(max_epochs):
        dw, db, num_points = 0, 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
            num_points += 1

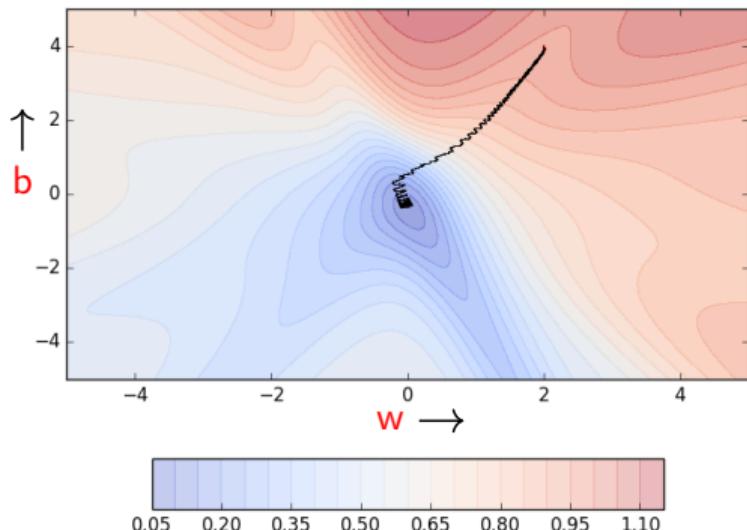
            if num_points_seen % mini_batch_size == 0:
                # seen one mini_batch
                w = w - eta * dw
                b = b - eta * db
                dw, db = 0, 0 #reset gradients
```

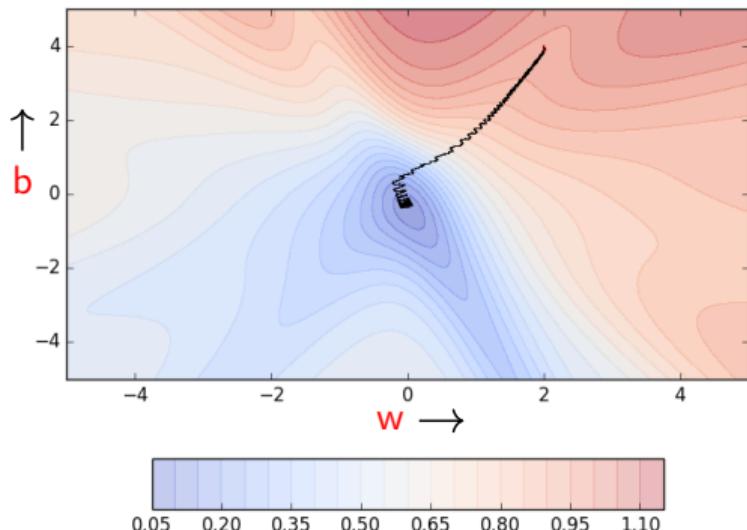
- 使用 *mini_batch_size* 个数据，对参数进行更新
- 比随机梯度下降只使用一个数据对参数更新会更好一些
- 看一个例子 ($k = 2$)

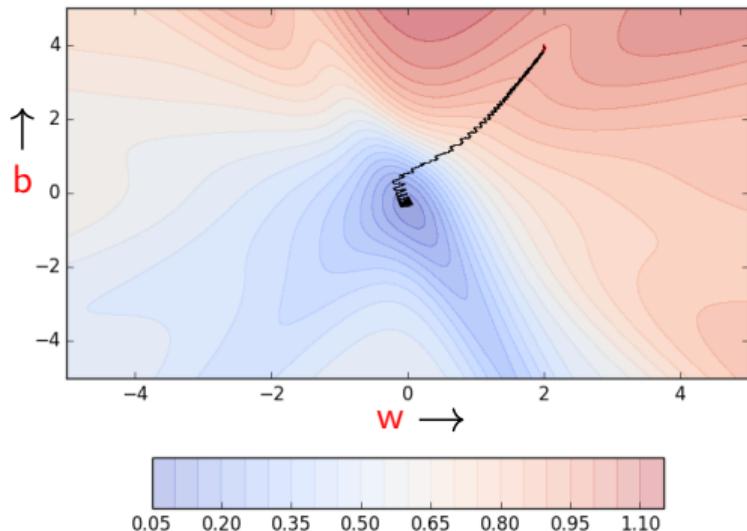
```
def do_stochastic_gradient_descent():
    w, b, eta, max_epochs = -2, -2, 1.0, 1000
    for i in range(max_epochs):
        dw, db = 0, 0
        for x, y in zip(X, Y):
            dw = grad_w(w, b, x, y)
            db = grad_b(w, b, x, y)
            w = w - eta * dw
            b = b - eta * db
```

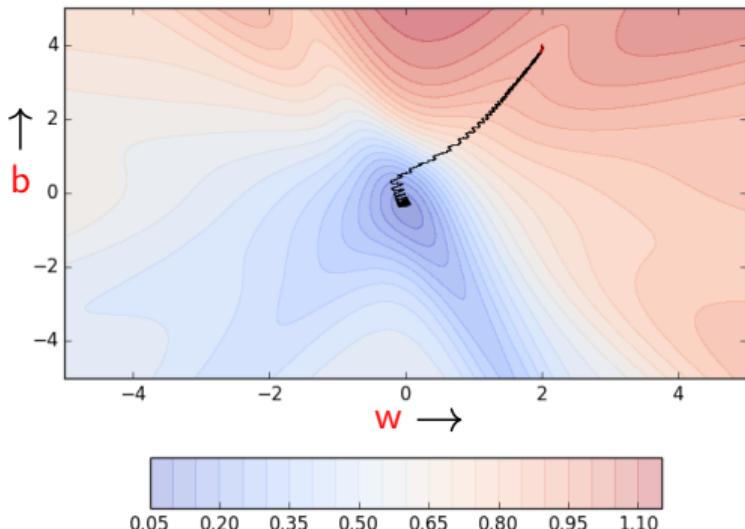


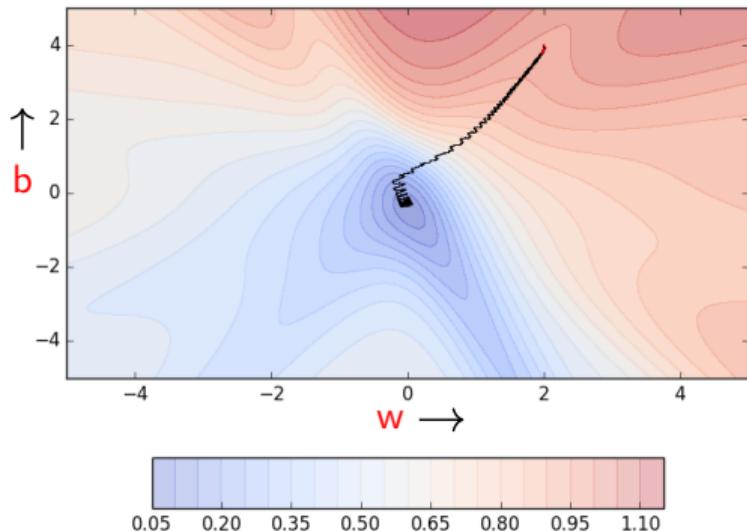


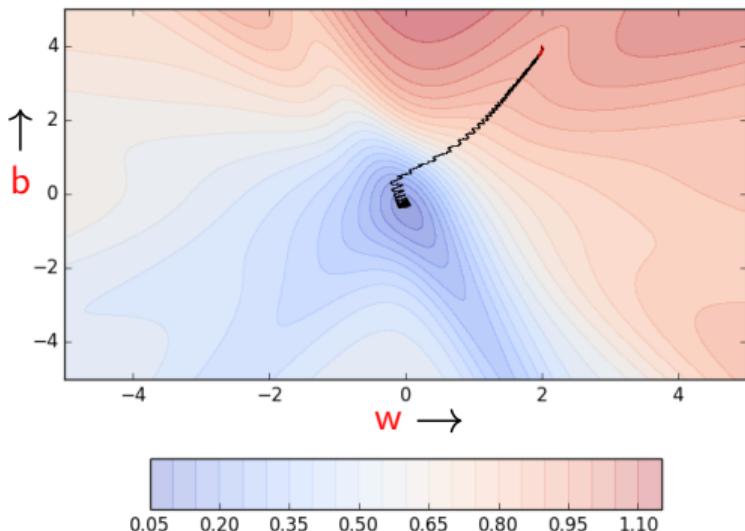


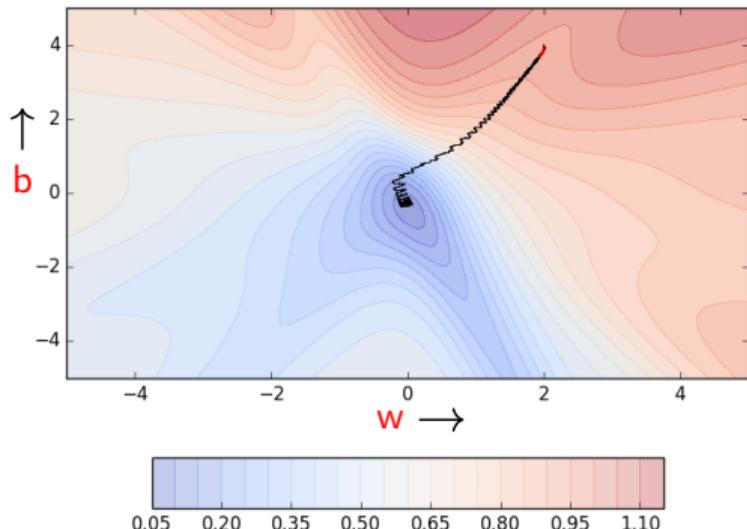


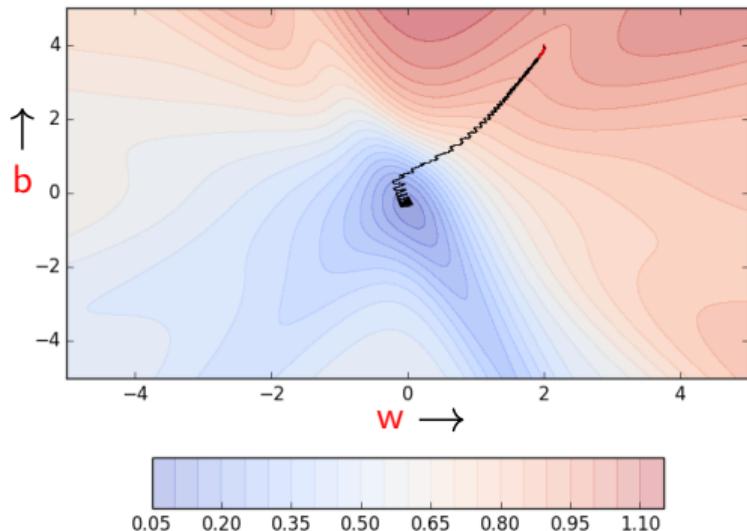


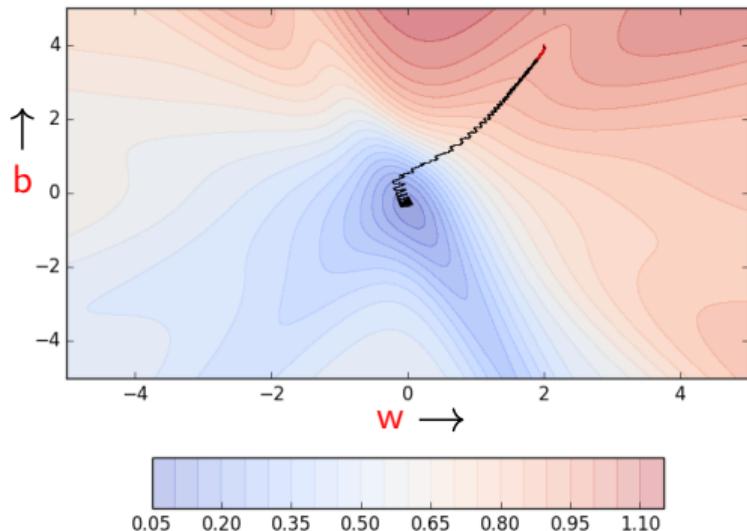


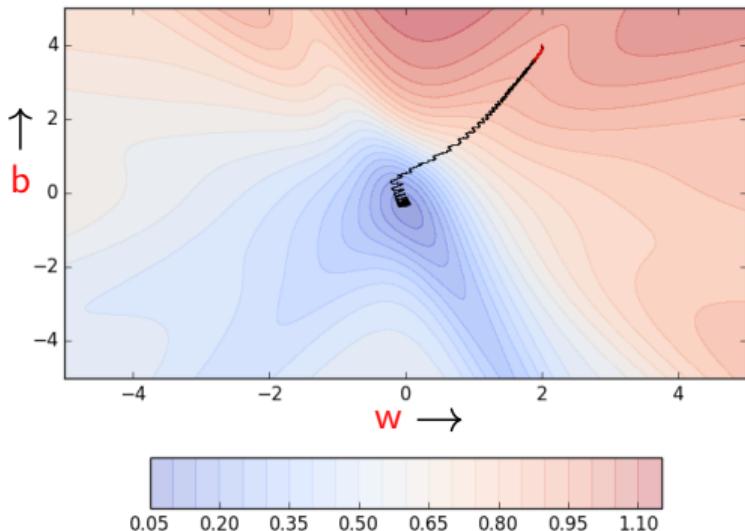


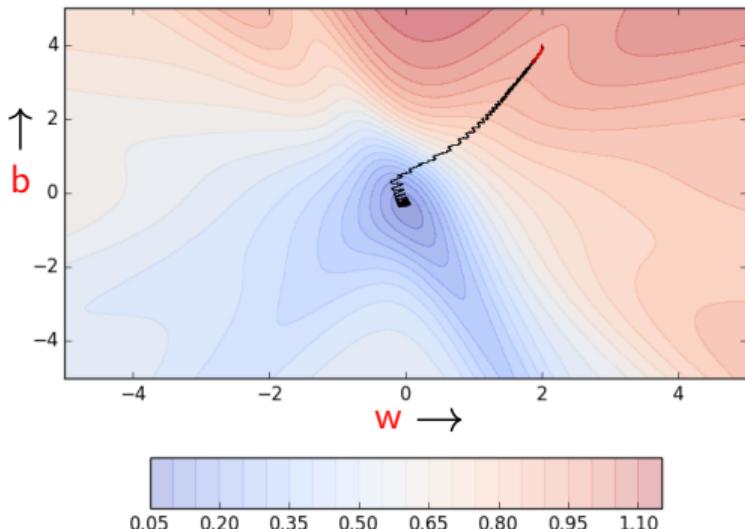


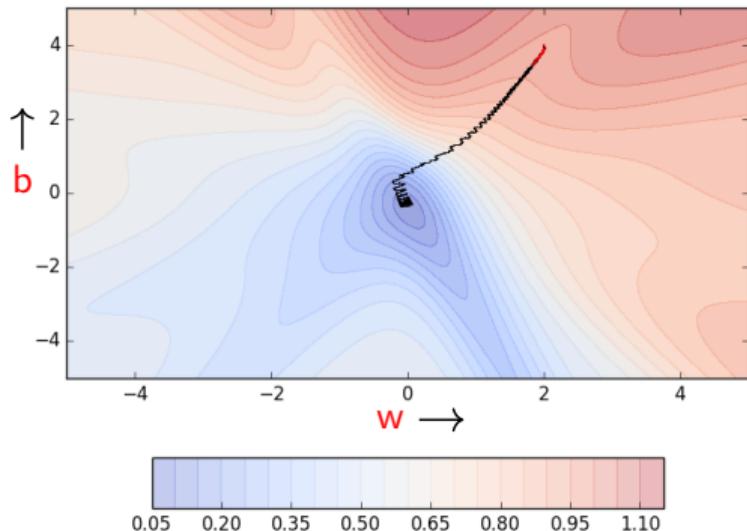


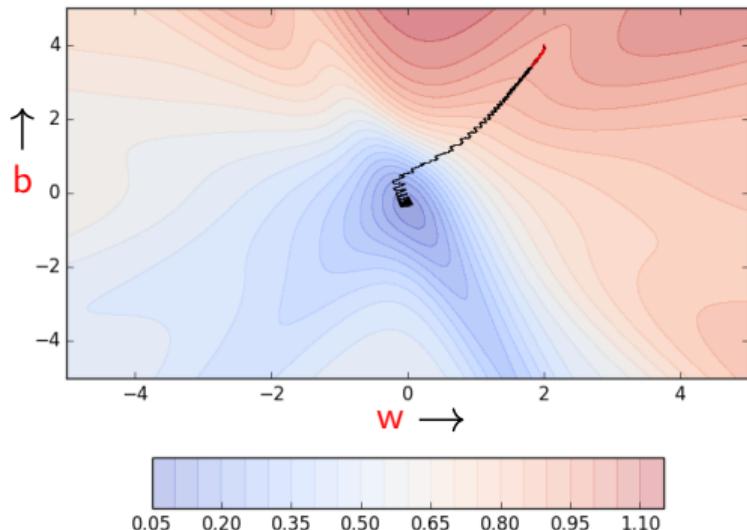


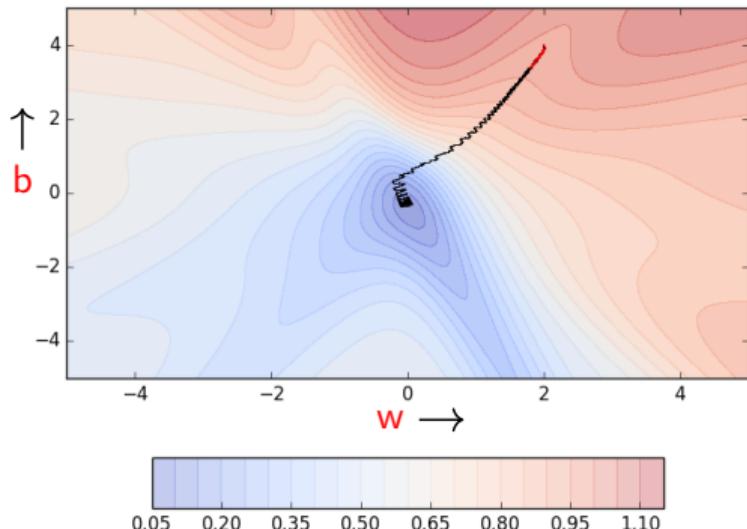


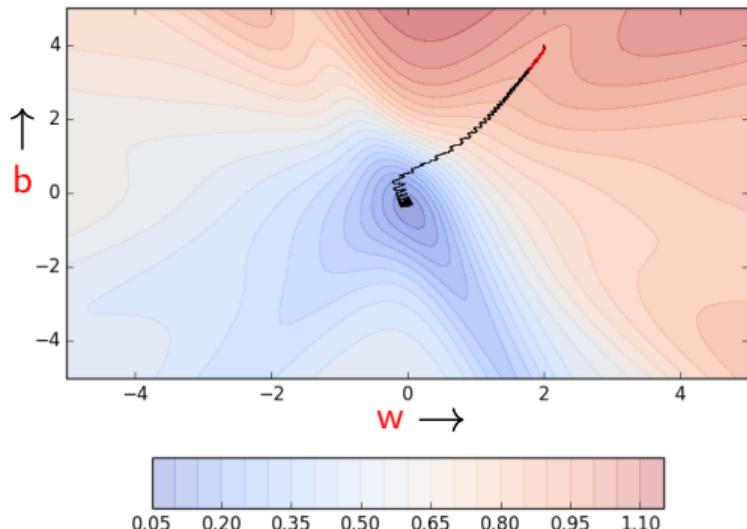


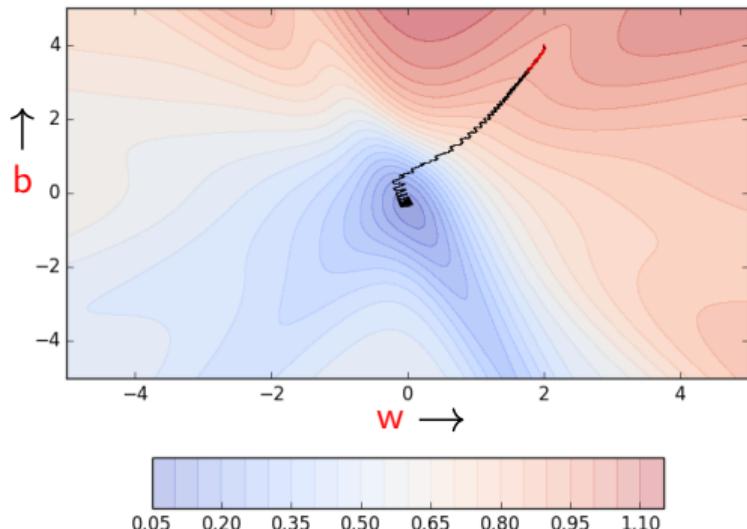


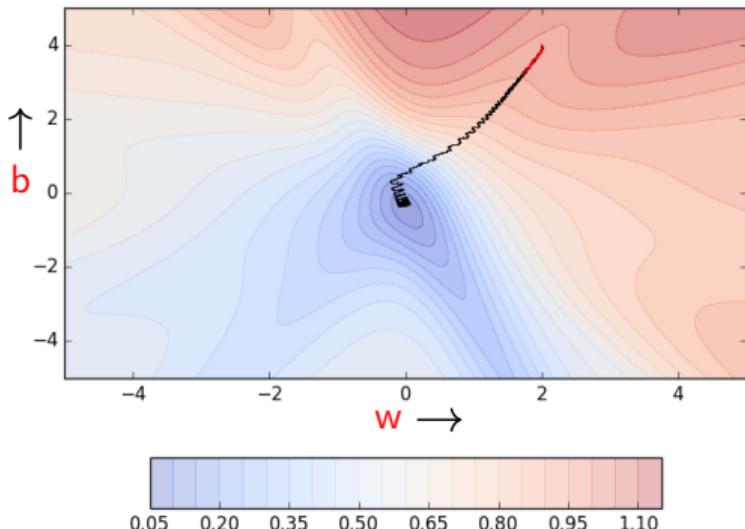


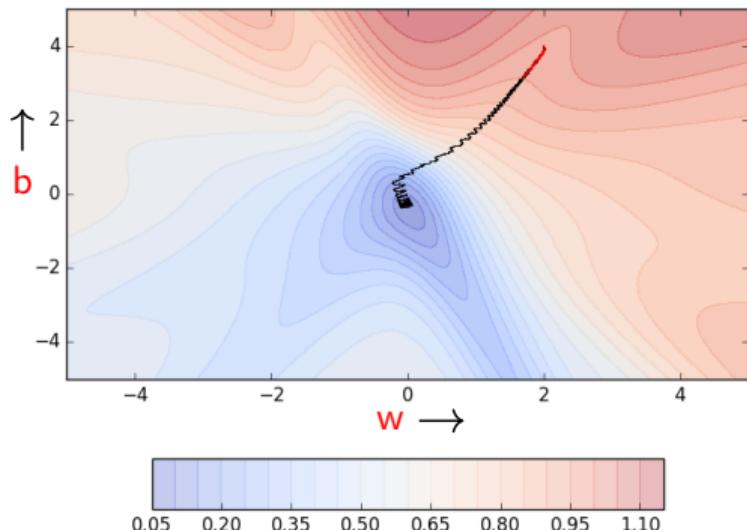


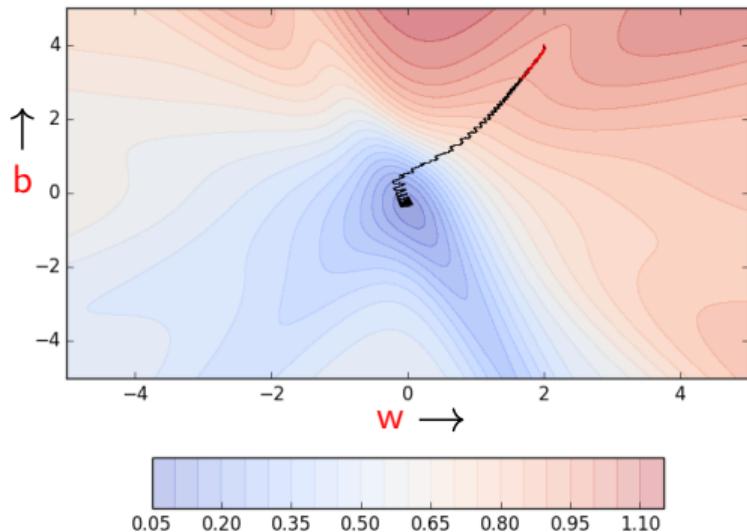


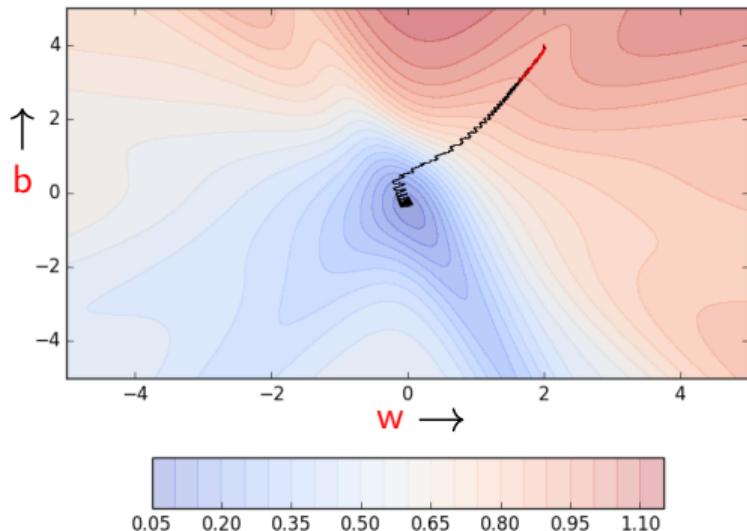


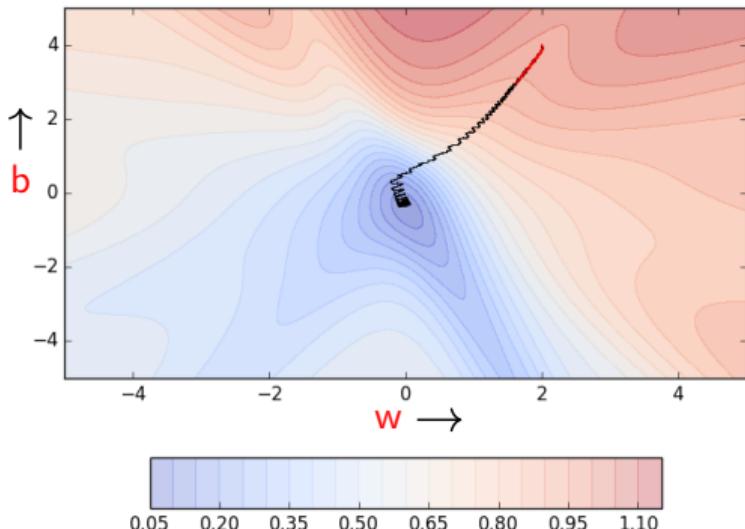


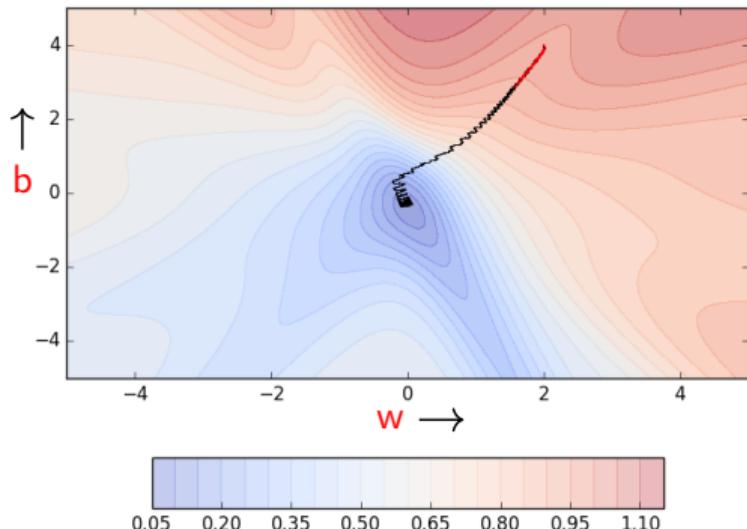


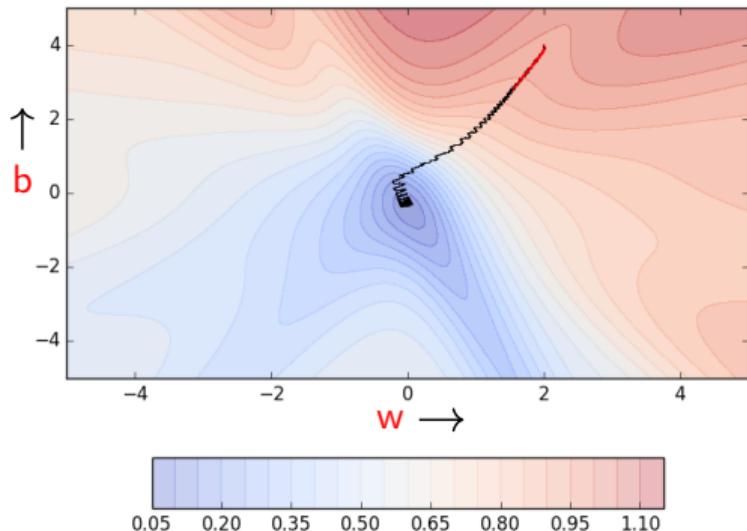


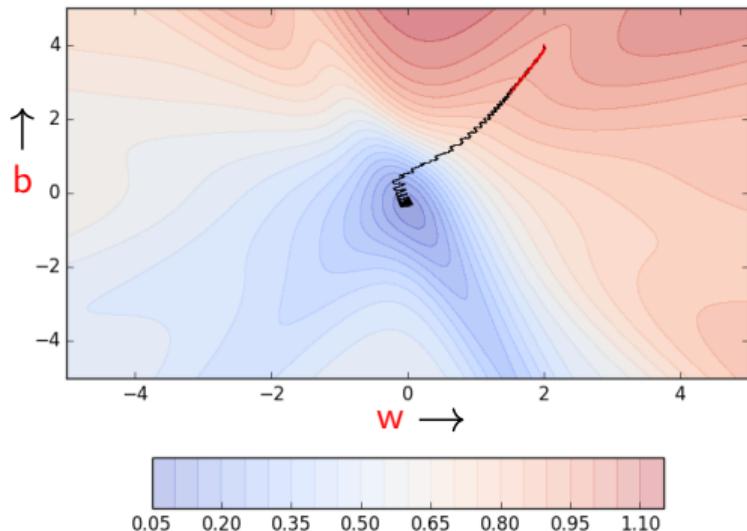


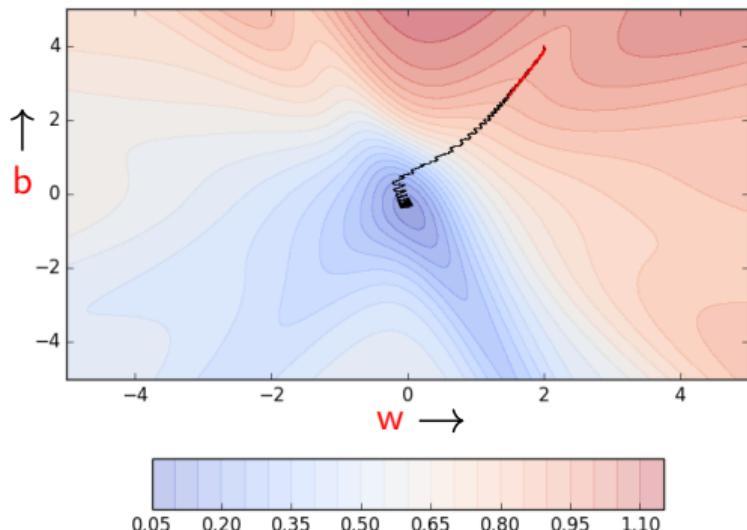


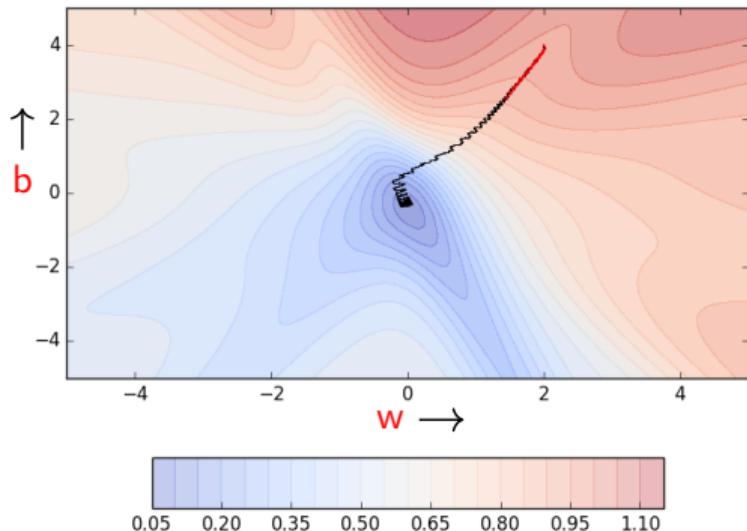


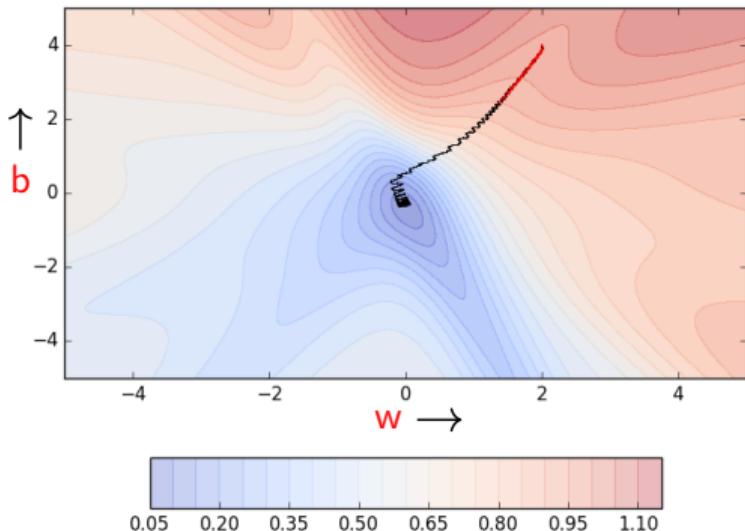


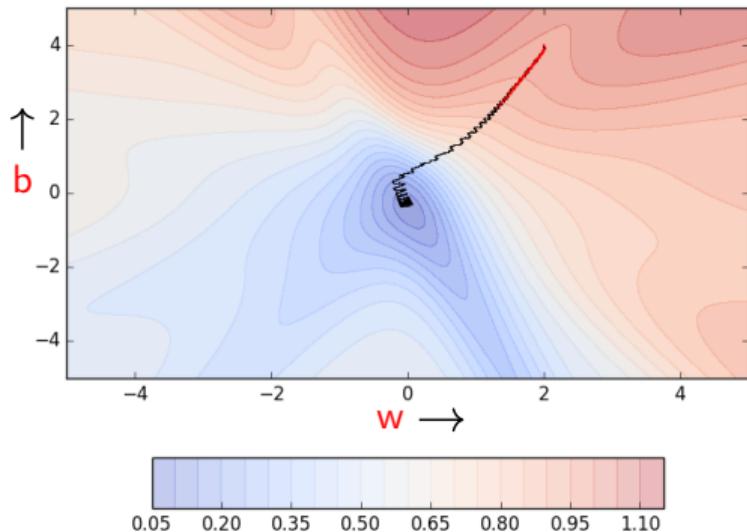


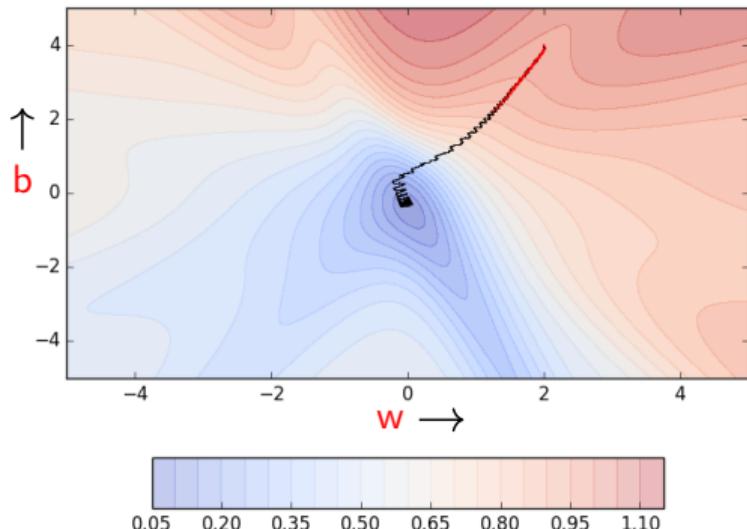


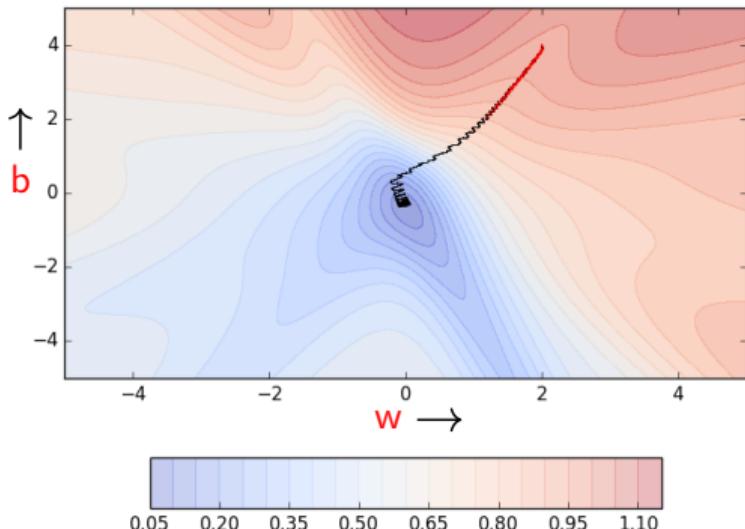


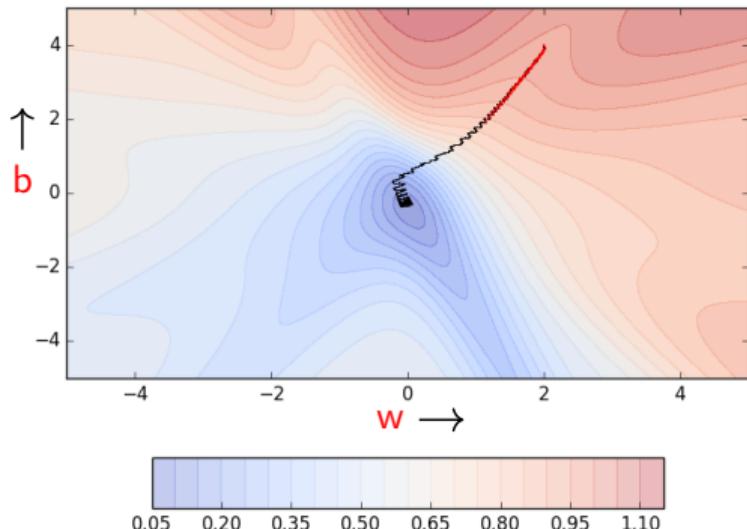


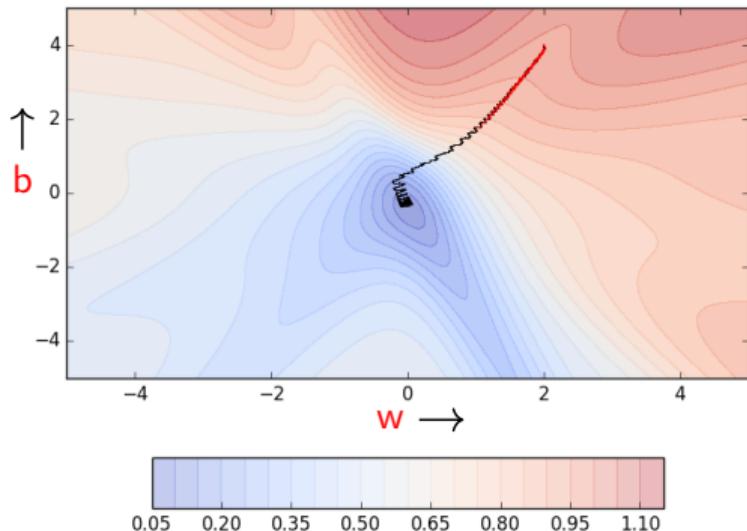


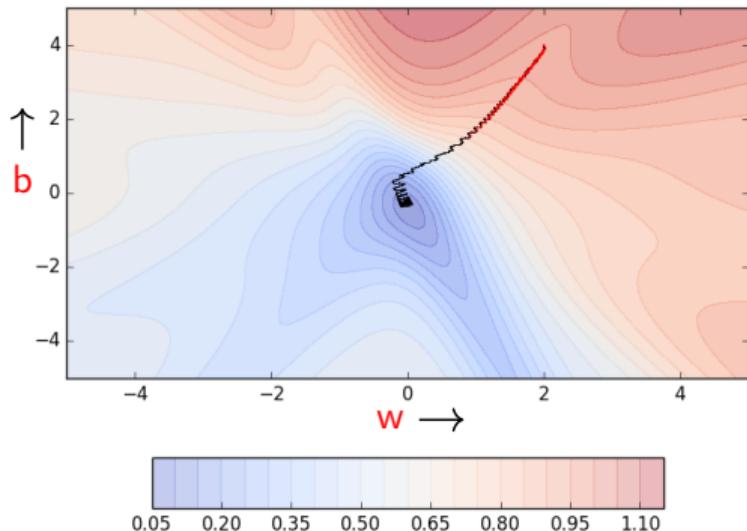


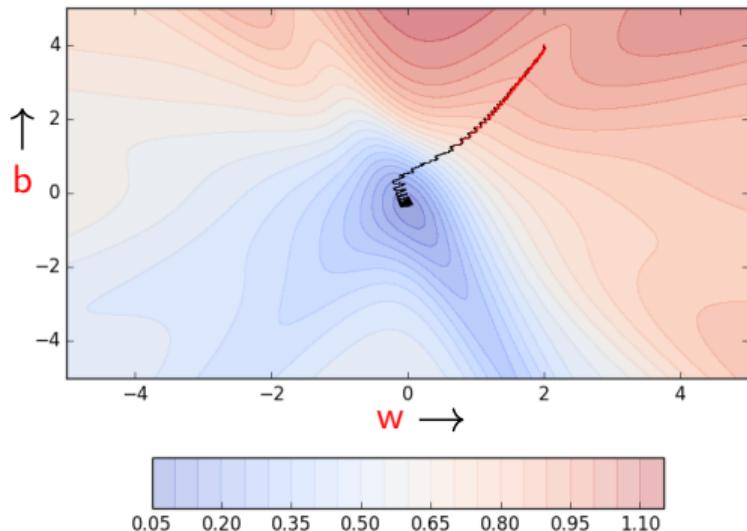


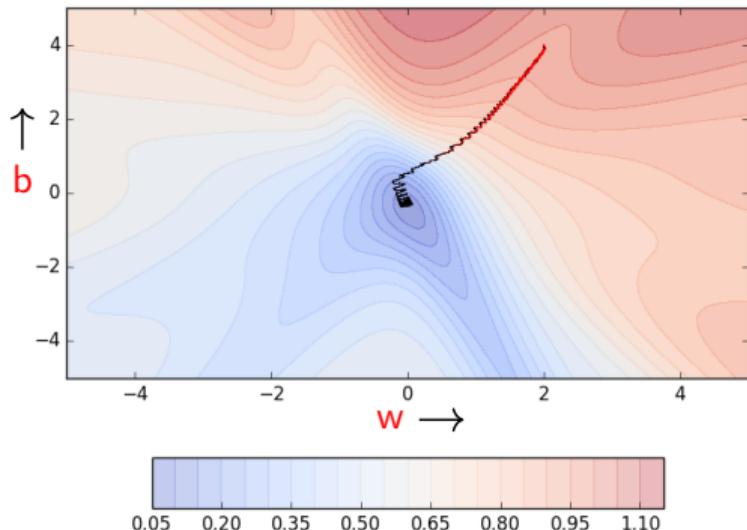


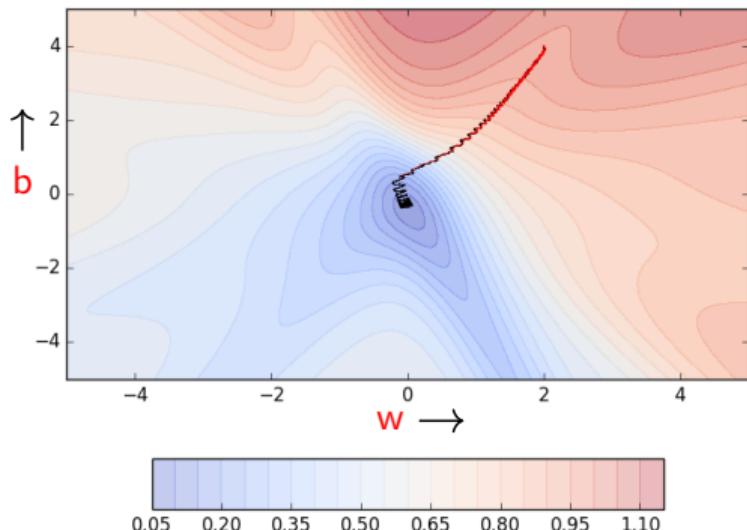


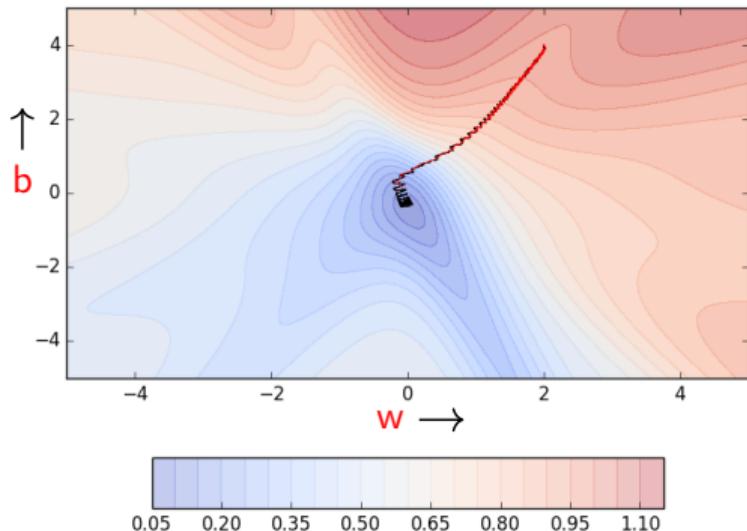


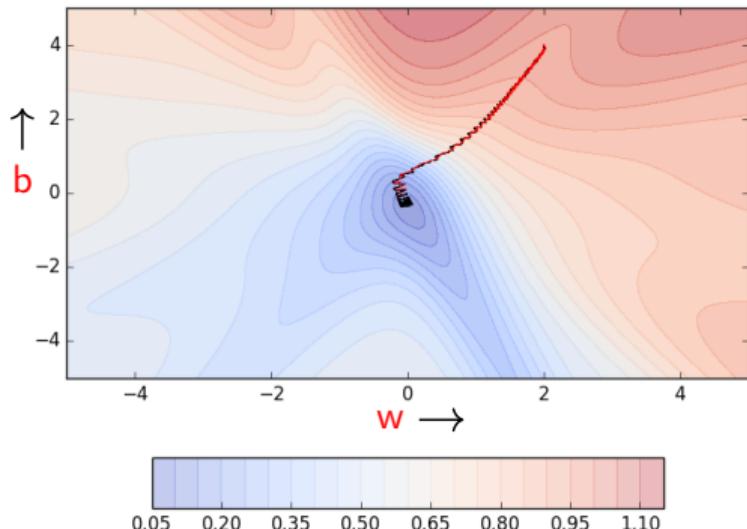


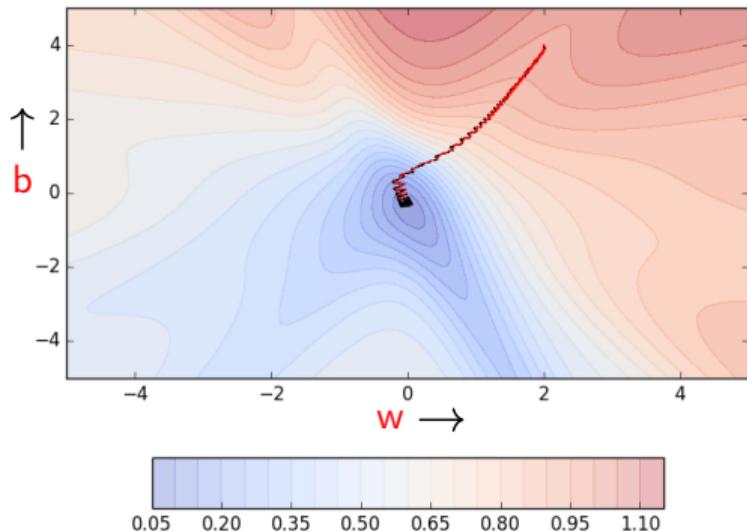


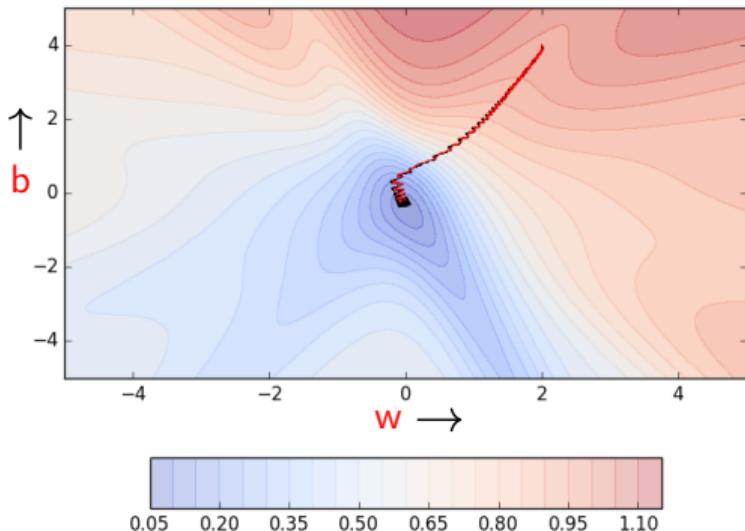


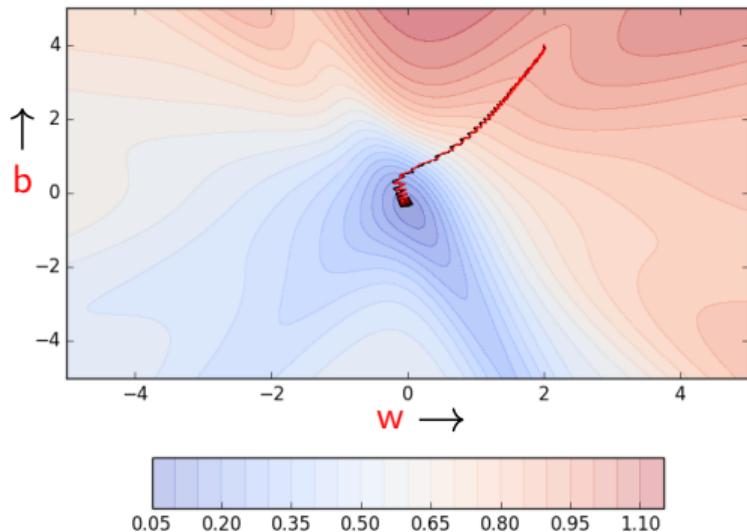


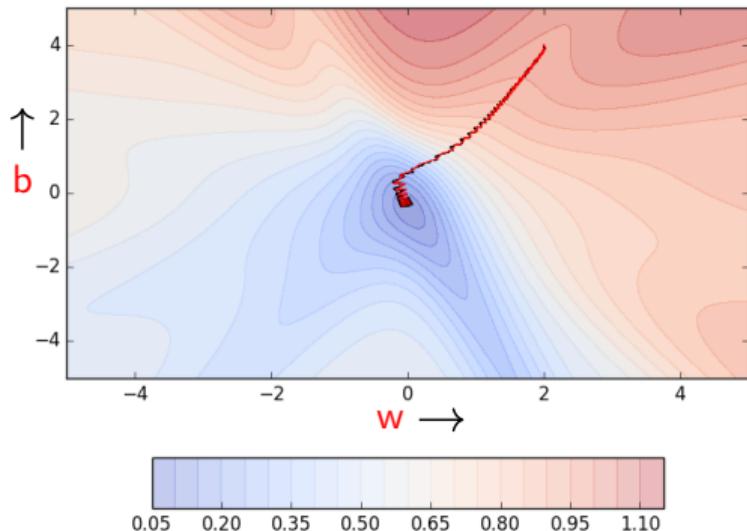




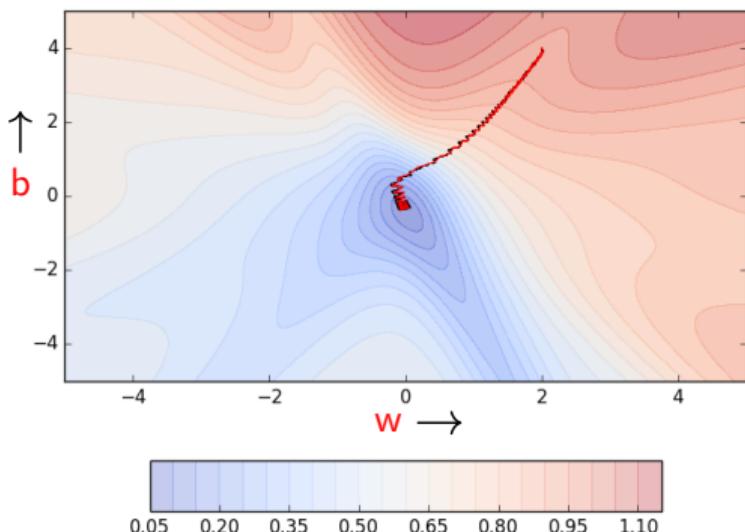




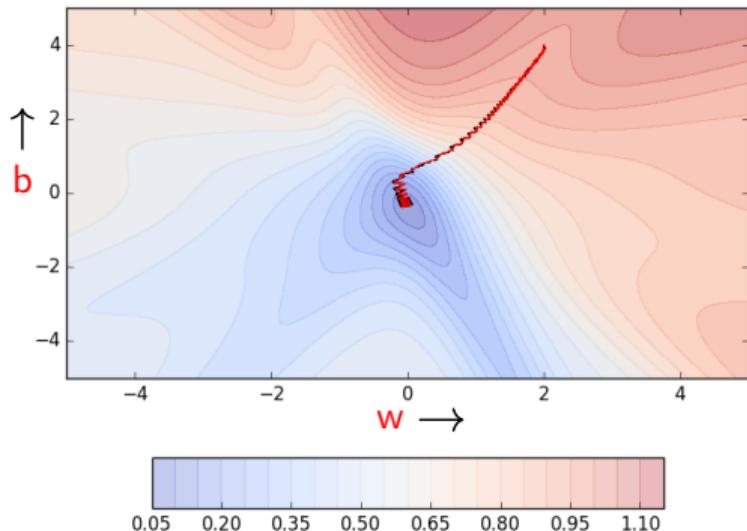




- 即使当批大小 $k = 2$ 时，振荡也明显下降

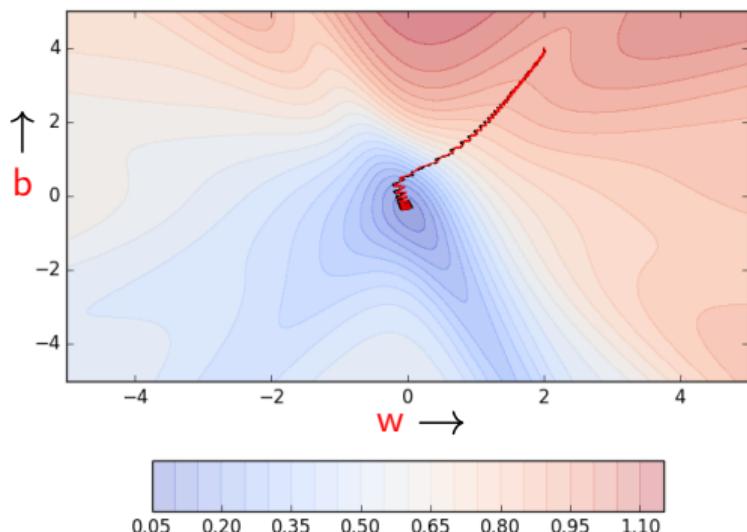


- 即使当批大小 $k = 2$ 时，振荡也明显下降
Why ?

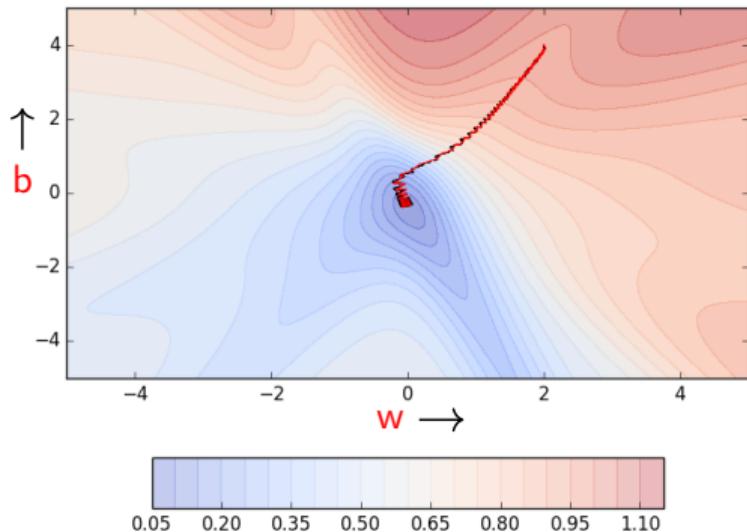




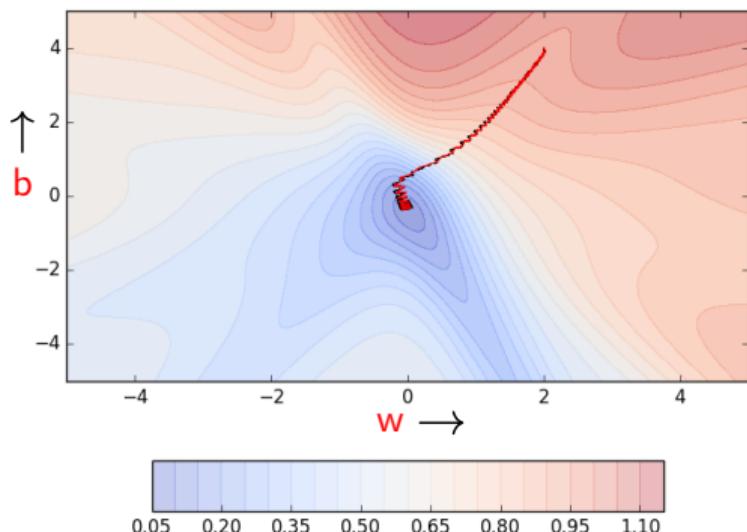
- 即使当批大小 $k = 2$ 时，振荡也明显下降
Why ?
- 因为对梯度的估计比之前更好一些



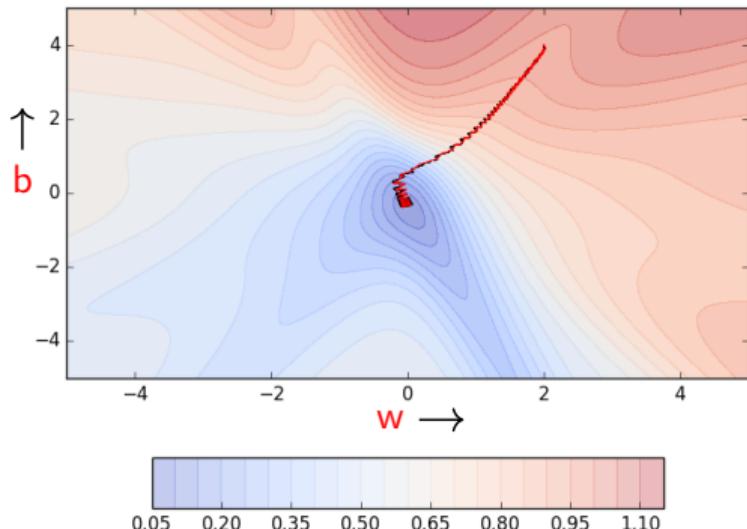
- 即使当批大小 $k = 2$ 时，振荡也明显下降
Why ?
- 因为对梯度的估计比之前更好一些 [类似于：抛硬币 $k=2$ 次来估计 $P(\text{heads})$]



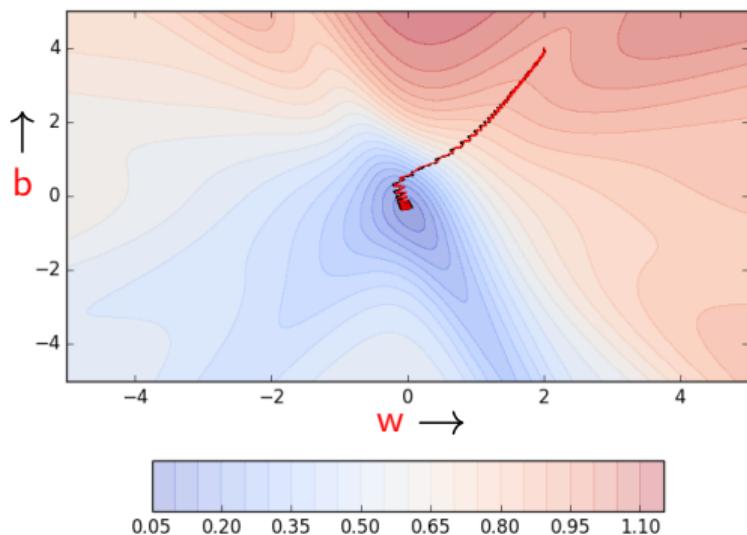
- 即使当批大小 $k = 2$ 时，振荡也明显下降
Why ?
- 因为对梯度的估计比之前更好一些 [类似于：抛硬币 $k=2$ 次来估计 $P(\text{heads})$]
- k 的值越大，对梯度的估计越好



- 即使当批大小 $k = 2$ 时，振荡也明显下降
Why ?
- 因为对梯度的估计比之前更好一些 [类似于：抛硬币 $k=2$ 次来估计 $P(\text{heads})$]
- k 的值越大，对梯度的估计越好
- 实际应用中， k 的值经常取为 16, 32, 64



- 即使当批大小 $k = 2$ 时，振荡也明显下降
Why ?
- 因为对梯度的估计比之前更好一些 [类似于：抛硬币 $k=2$ 次来估计 $P(\text{heads})$]
- k 的值越大，对梯度的估计越好
- 实际应用中， k 的值经常取为 16, 32, 64
- 当然，振荡仍然存在。只要使用部分数据进行梯度估计，振荡就会存在





三种梯度下降方法的比较

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	N
Stochastic Gradient Descent	N
Mini-Batch Gradient Descent	$\frac{N}{B}$



三种梯度下降方法的比较

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	1
Stochastic Gradient Descent	
Mini-Batch Gradient Descent	



三种梯度下降方法的比较

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	1
Stochastic Gradient Descent	N
Mini-Batch Gradient Descent	



三种梯度下降方法的比较

- 1 epoch = one pass over the entire data
- 1 step = one update of the parameters
- N = number of data points
- B = Mini batch size

Algorithm	# of steps in 1 epoch
Vanilla (Batch) Gradient Descent	1
Stochastic Gradient Descent	N
Mini-Batch Gradient Descent	$\frac{N}{B}$



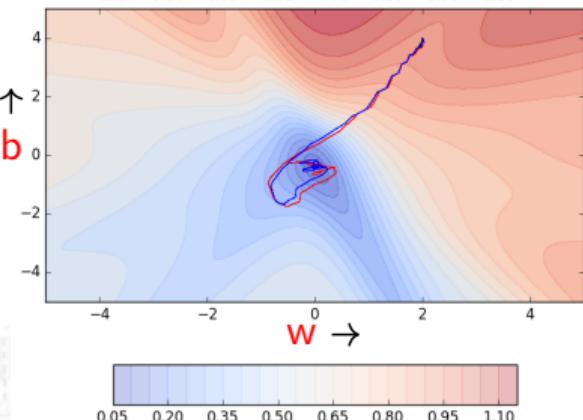
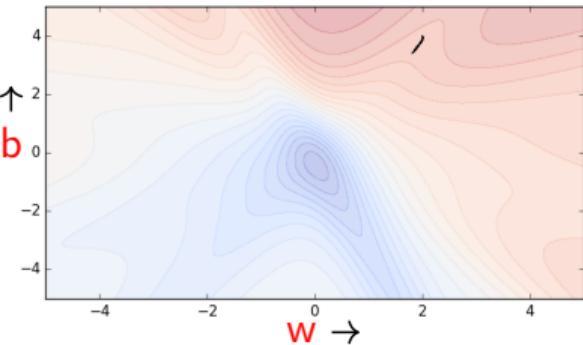
大家写写 *stochastic versions of Momentum based gradient descent and Nesterov accelerated based gradient descent*

```
def do_momentum_gradient_descent() :  
    w, b, eta = init_w, init_b, 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 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 = gamma * prev_v_w + eta* dw  
            v_b = gamma * prev_v_b + eta* db  
            w = w - v_w  
            b = b - v_b  
            prev_v_w = v_w  
            prev_v_b = v_b
```

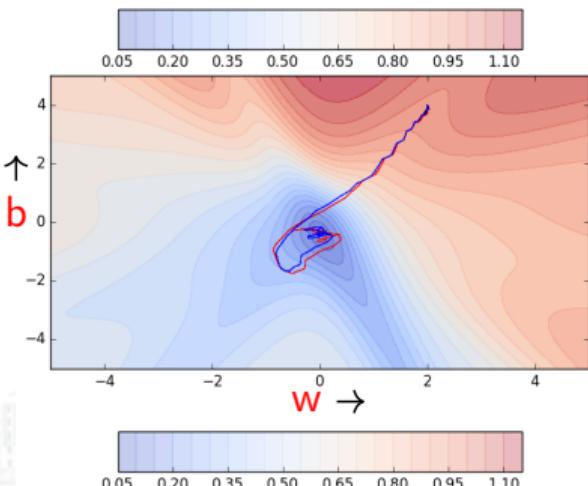
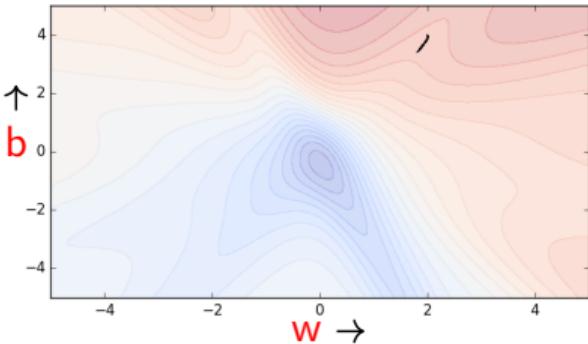
```
def do_stochastic_momentum_gradient_descent() :  
    w, b, eta = init_w, init_b, 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        for x,y in zip(X, Y) :  
            dw = grad_w(w, b, x, y)  
            db = grad_b(w, b, x, y)  
  
            v_w = gamma * prev_v_w + eta* dw  
            v_b = gamma * prev_v_b + eta* db  
            w = w - v_w  
            b = b - v_b  
            prev_v_w = v_w  
            prev_v_b = v_b
```

```
def do_nesterov_accelerated_gradient_descent() :  
  
    w, b, eta = init_w, init_b , 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        #do partial updates  
        v_w = gamma * prev_v_w  
        v_b = gamma * prev_v_b  
        for x,y in zip(X, Y) :  
            #calculate gradients after partial update  
            dw += grad_w(w - v_w, b - v_b, x, y)  
            db += grad_b(w - v_w, b - v_b, x, y)  
  
        #now do the full update  
        v_w = gamma * prev_v_w + eta * dw  
        v_b = gamma * prev_v_b + eta * db  
        w = w - v_w  
        b = b - v_b  
        prev_v_w = v_w  
        prev_v_b = v_b
```

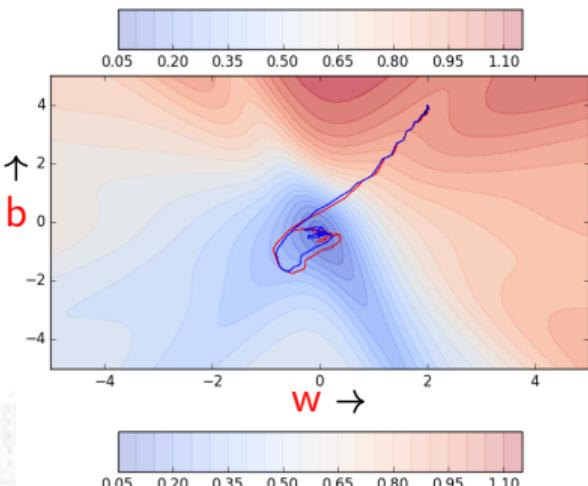
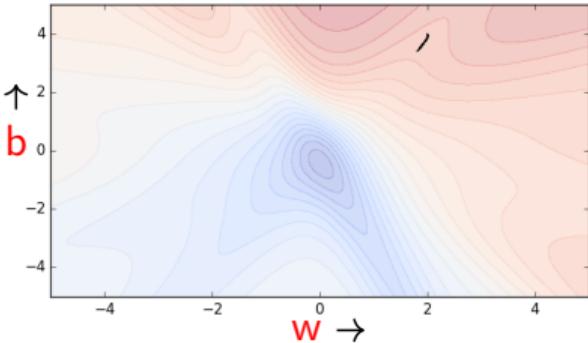
```
def do_nesterov_accelerated_gradient_descent() :  
    w, b, eta = init_w, init_b, 1.0  
    prev_v_w, prev_v_b, gamma = 0, 0, 0.9  
    for i in range(max_epochs) :  
        dw, db = 0, 0  
        for x,y in zip(X, Y) :  
            #do partial updates  
            v_w = gamma * prev_v_w  
            v_b = gamma * prev_v_b  
            #calculate gradients after partial update  
            dw = grad_w(w - v_w, b - v_b, x, y)  
            db = grad_b(w - v_w, b - v_b, x, y)  
  
            v_w = gamma * prev_v_w + eta * dw  
            v_b = gamma * prev_v_b + eta * db  
            w = w - v_w  
            b = b - v_b  
            prev_v_w = v_w  
            prev_v_b = v_b
```



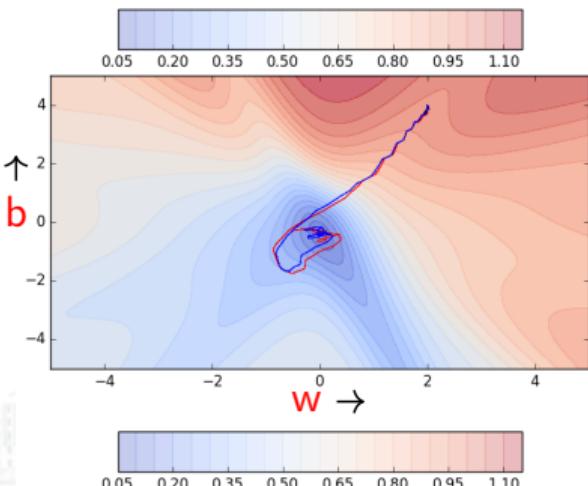
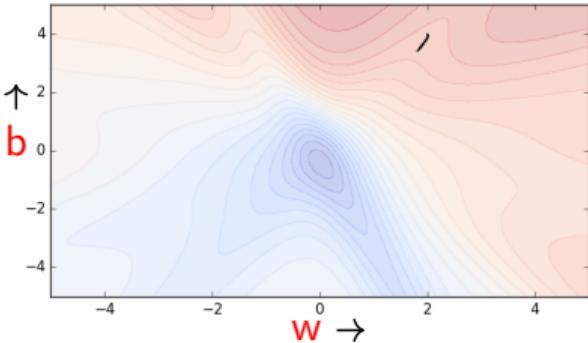
- 虽然 Momentum [红色] 和 NAG [蓝色] 的随机版本都表现出振荡，但 NAG 相对于 Momentum 的相对优势仍然成立



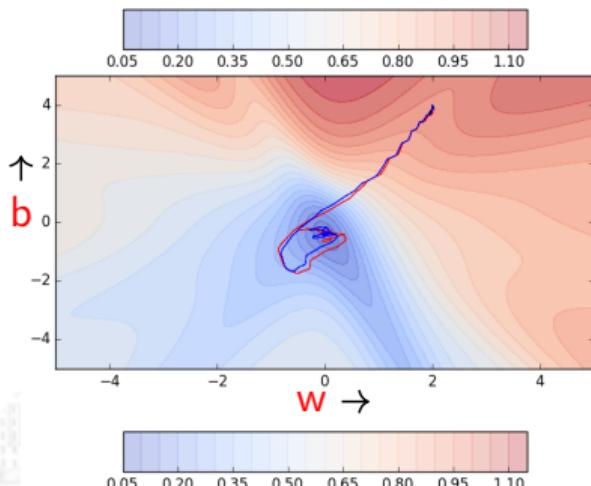
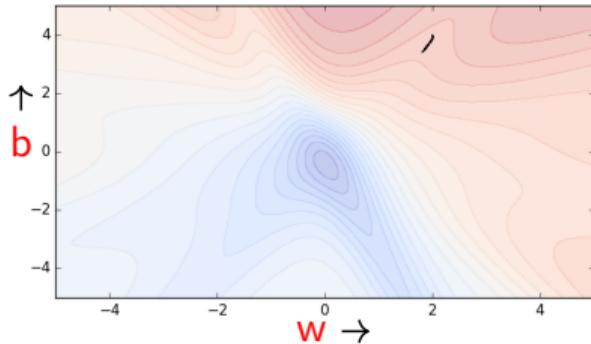
- 虽然 Momentum [红色] 和 NAG [蓝色] 的随机版本都表现出振荡，但 NAG 相对于 Momentum 的相对优势仍然成立 (i.e., NAG takes relatively shorter u-turns)



- 虽然 Momentum [红色] 和 NAG [蓝色] 的随机版本都表现出振荡，但 NAG 相对于 Momentum 的相对优势仍然成立 (i.e., NAG takes relatively shorter u-turns)
- 它们都比随机梯度下降要快



- 虽然 Momentum [红色] 和 NAG [蓝色] 的随机版本都表现出振荡, 但 NAG 相对于 Momentum 的相对优势仍然成立 (i.e., NAG takes relatively shorter u-turns)
- 它们都比随机梯度下降要快 (60 次更新后, stochastic gradient descent (上图中的黑色) 仍然有较大的损失, 但 NAG and Momentum 已经趋近于收敛)





写写 *the mini batch version of Momentum and NAG*



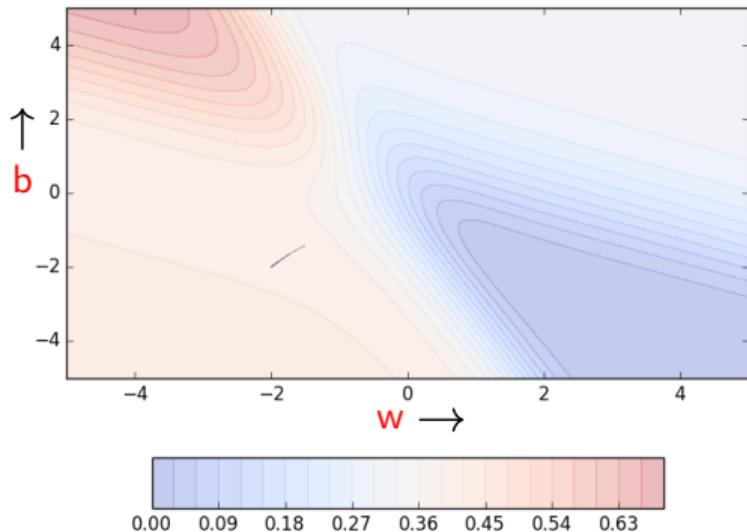
调整学习率和动量的技巧



学习率应该设置的大还是小？

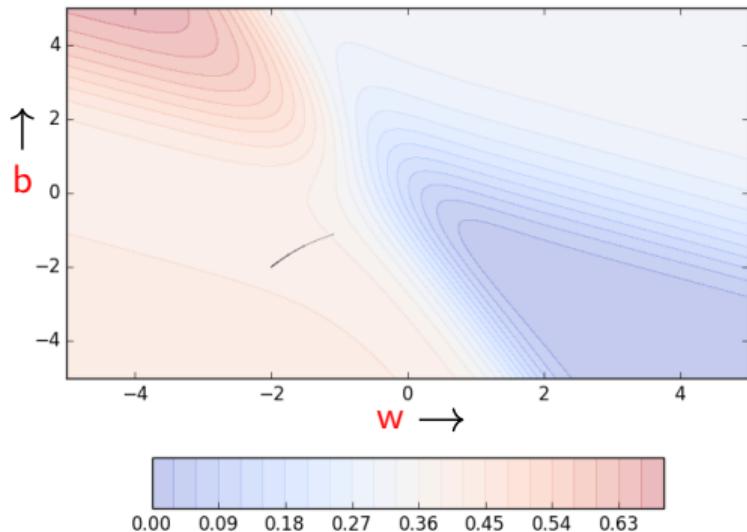


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）



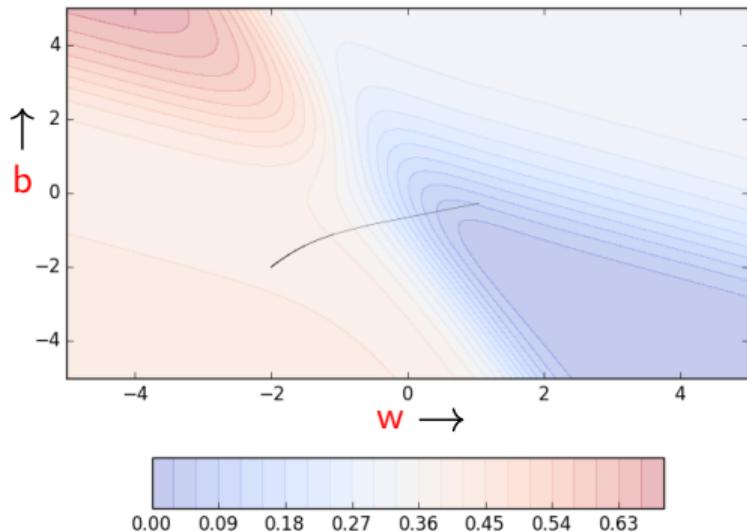


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



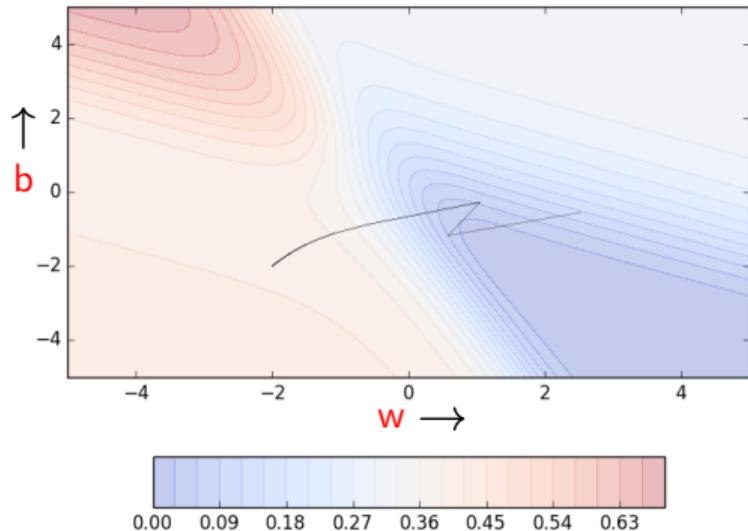


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



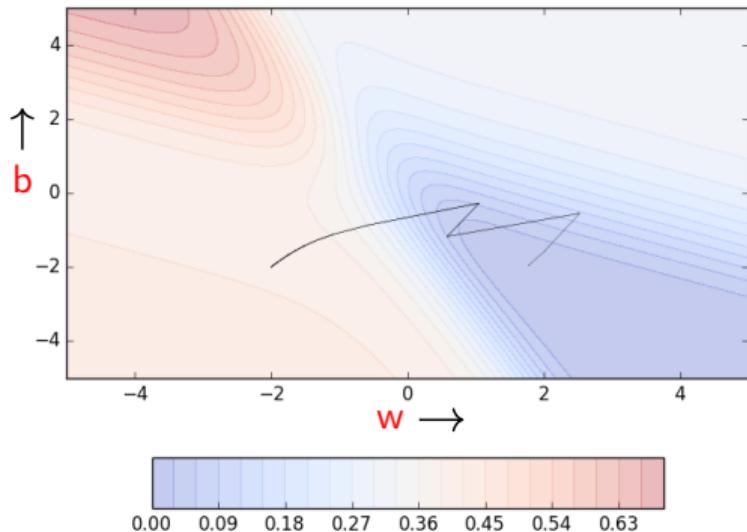


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



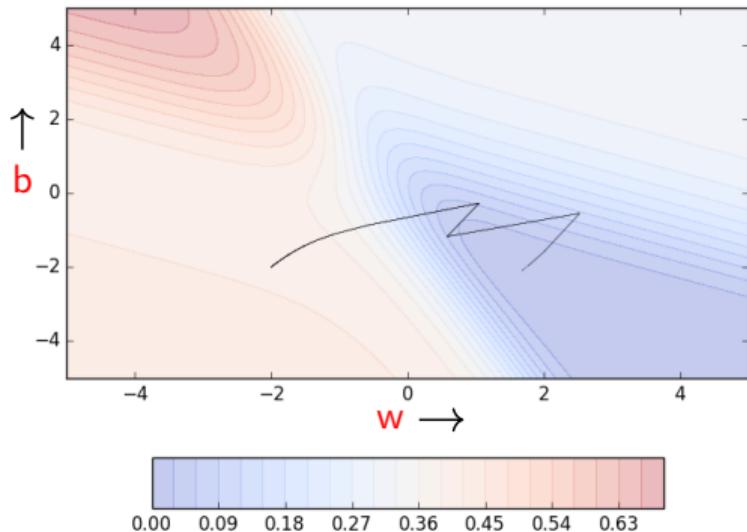


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



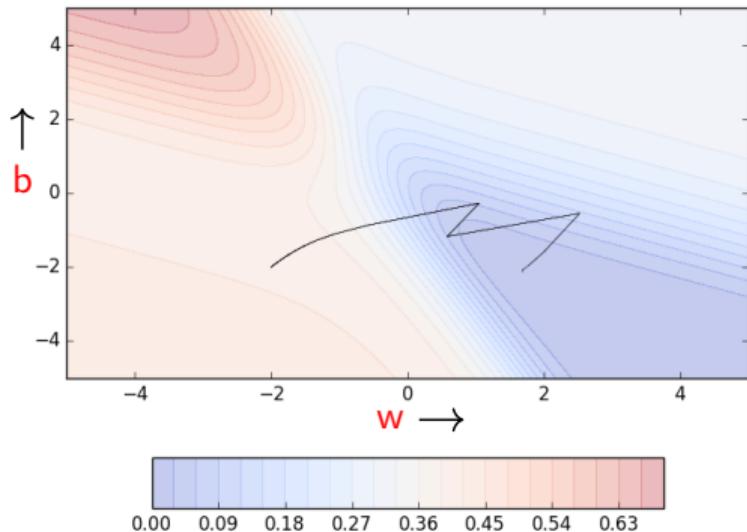


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



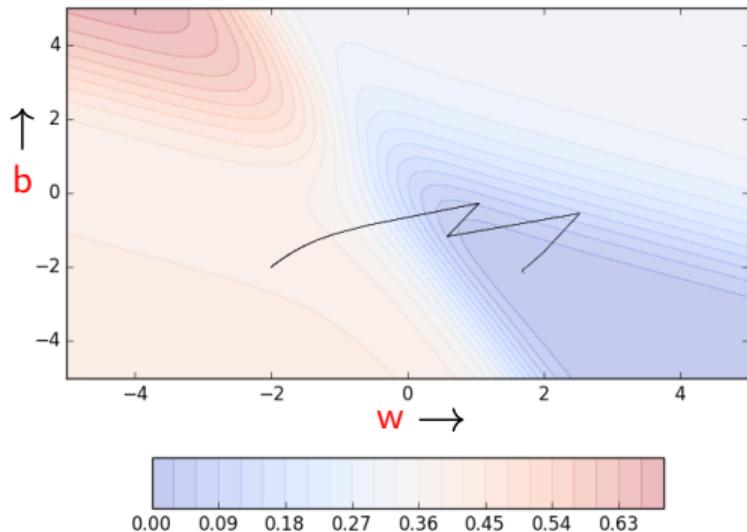


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



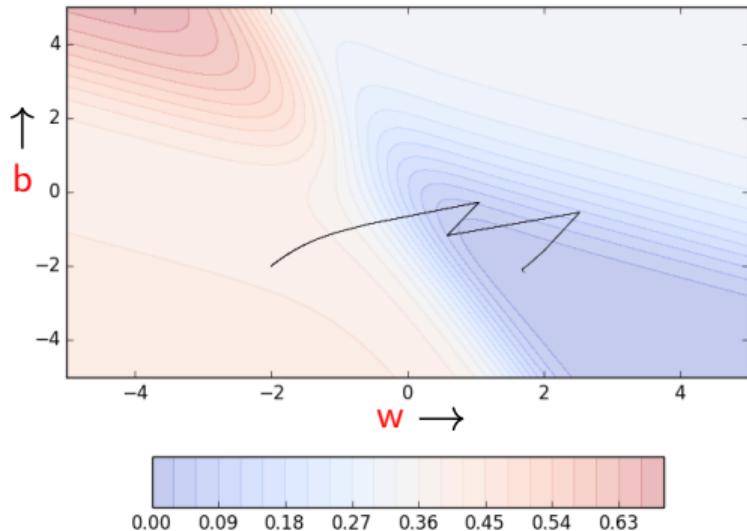


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



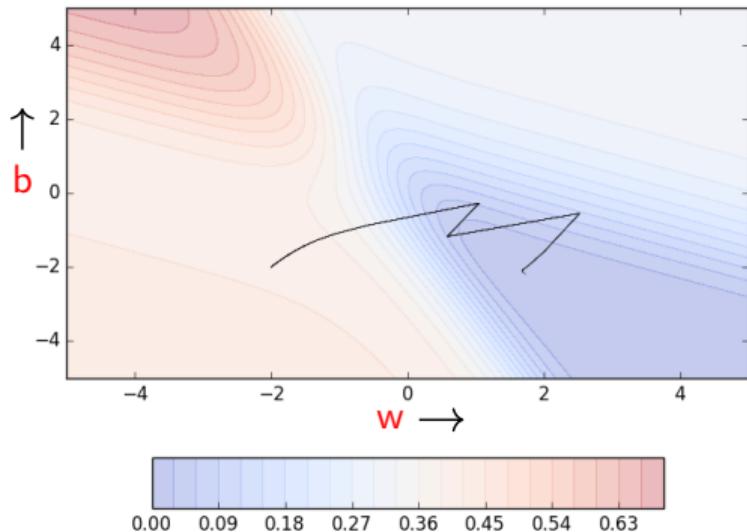


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



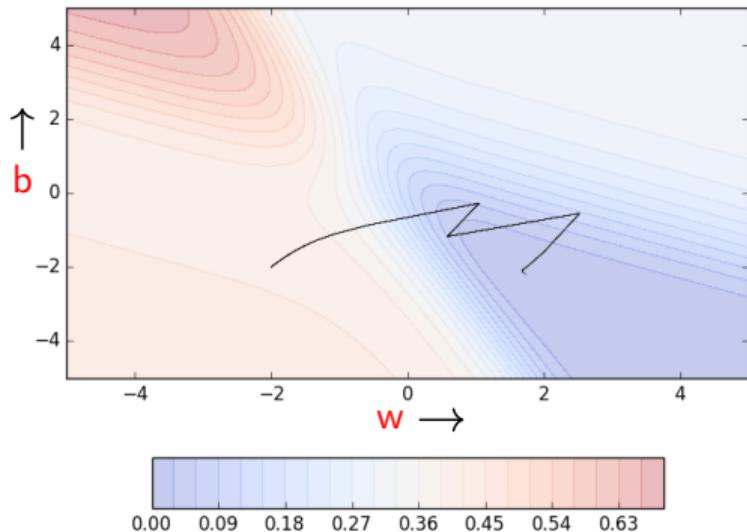


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



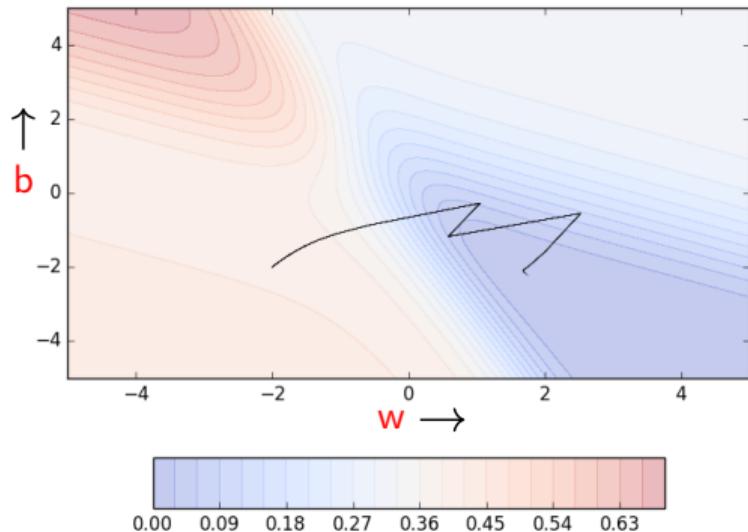


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



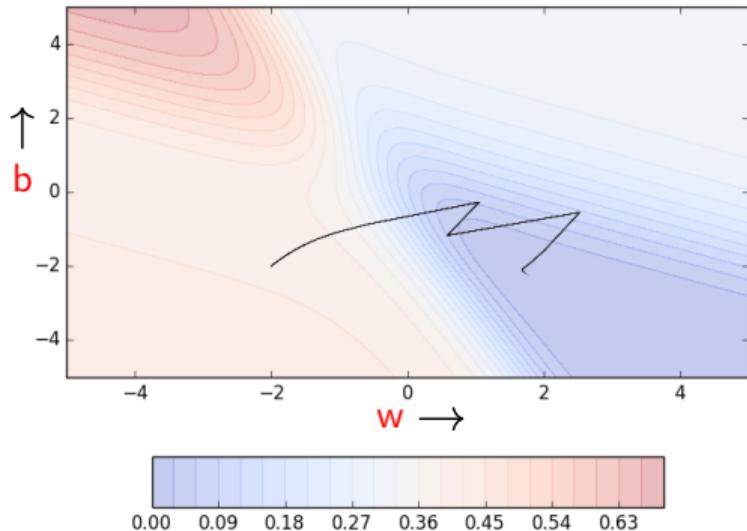


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



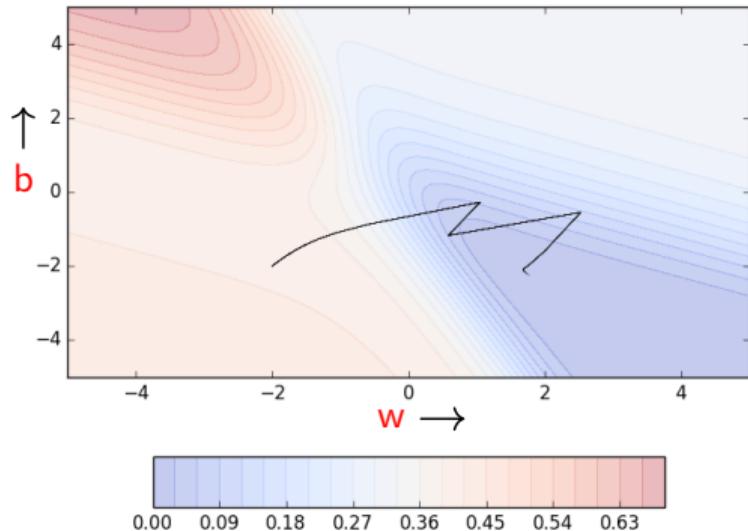


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



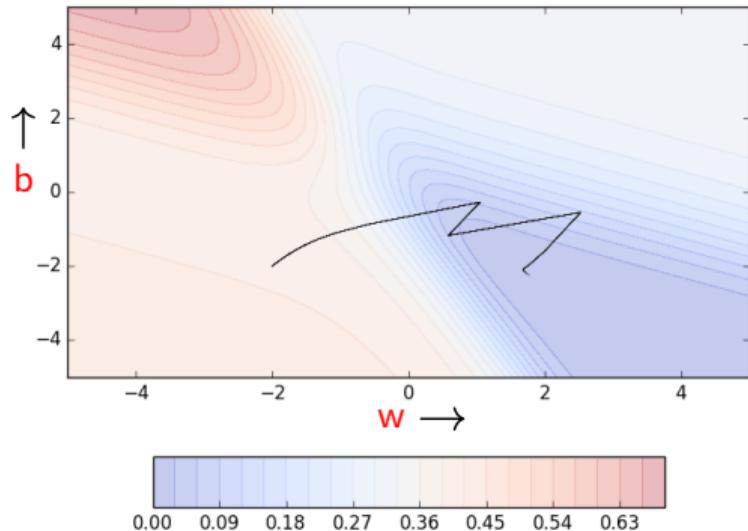


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



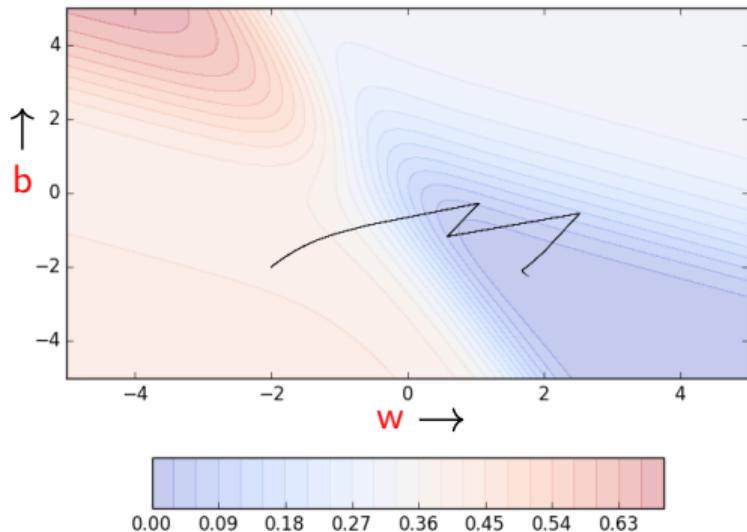


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



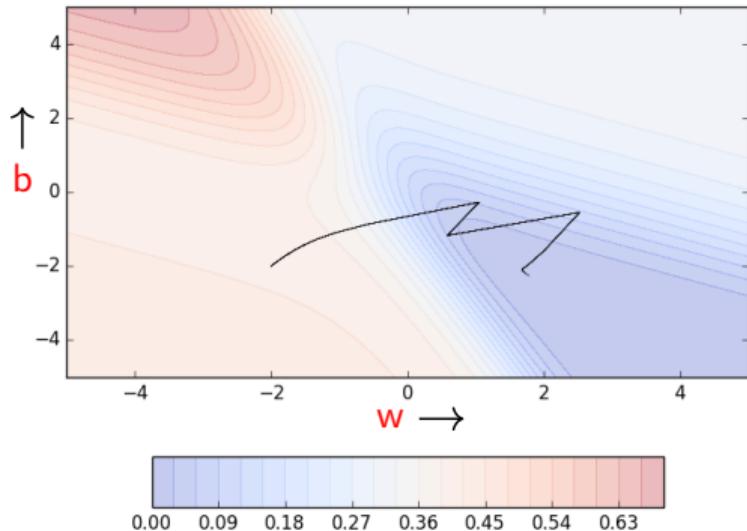


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



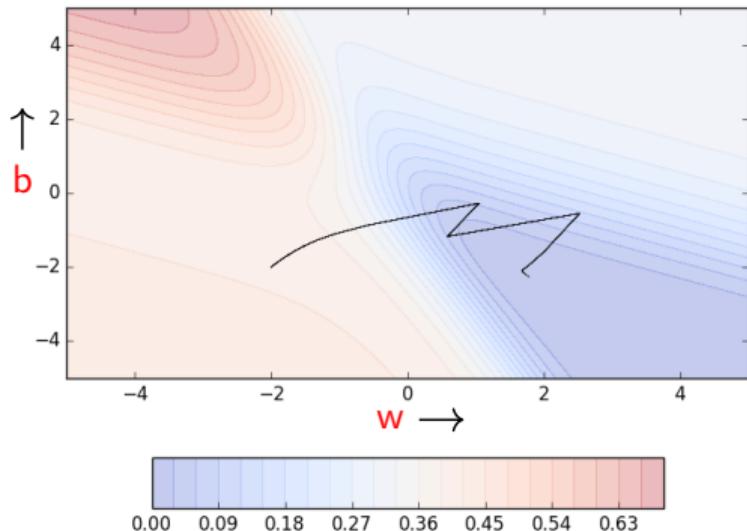


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



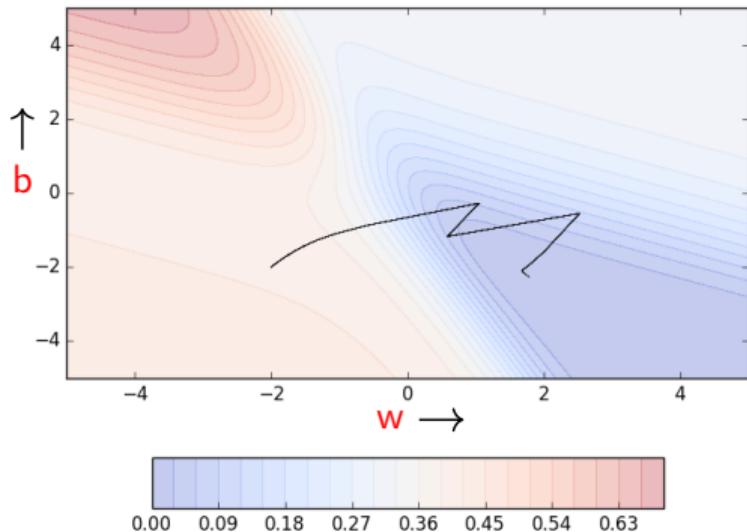


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？



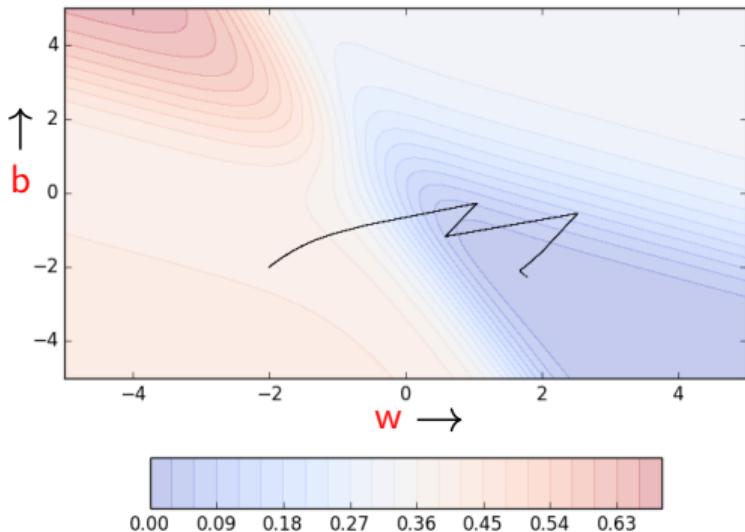


- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？
- 在陡峭的区域，梯度已经很大了，乘以一个高的学习率，会让更新的步伐更大，也会导致振荡





- 在平坦的区域，希望更快一些，是否可以通过设置一个较高的学习率来实现？）
- 看看当学习率设为 10 时会发生什么？
- 在陡峭的区域，梯度已经很大了，乘以一个高的学习率，会让更新的步伐更大，也会导致振荡
- 理想情况下，希望根据梯度的大小来调整学习率





Tips for initial learning rate ?



Tips for initial learning rate ?

- Tune learning rate [Try different values on a log scale: 0.0001, 0.001, 0.01, 0.1, 1.0]



Tips for initial learning rate ?

- Tune learning rate [Try different values on a log scale: 0.0001, 0.001, 0.01, 0.1, 1.0]
- Run a few epochs with each of these and figure out a learning rate which works best

Tips for initial learning rate ?

- Tune learning rate [Try different values on a log scale: 0.0001, 0.001, 0.01, 0.1, 1.0]
- Run a few epochs with each of these and figure out a learning rate which works best
- Now do a finer search around this value [for example, if the best learning rate was 0.1 then now try some values around it: 0.05, 0.2, 0.3]



Tips for initial learning rate ?

- Tune learning rate [Try different values on a log scale: 0.0001, 0.001, 0.01, 0.1, 1.0]
- Run a few epochs with each of these and figure out a learning rate which works best
- Now do a finer search around this value [for example, if the best learning rate was 0.1 then now try some values around it: 0.05, 0.2, 0.3]
- Disclaimer: these are just heuristics ... no clear winner strategy



Tips for annealing learning rate



Tips for annealing learning rate

- Step Decay:



Tips for annealing learning rate

- **Step Decay:**
 - Halve the learning rate after every 5 epochs or



Tips for annealing learning rate

- **Step Decay:**

- Halve the learning rate after every 5 epochs or
- Halve the learning rate after an epoch if the validation error is more than what it was at the end of the previous epoch

Tips for annealing learning rate

- **Step Decay:**
 - Halve the learning rate after every 5 epochs or
 - Halve the learning rate after an epoch if the validation error is more than what it was at the end of the previous epoch
- **Exponential Decay:** $\eta = \eta_0^{-kt}$ where η_0 and k are hyperparameters and t is the step number

Tips for annealing learning rate

- **Step Decay:**
 - Halve the learning rate after every 5 epochs or
 - Halve the learning rate after an epoch if the validation error is more than what it was at the end of the previous epoch
- **Exponential Decay:** $\eta = \eta_0^{-kt}$ where η_0 and k are hyperparameters and t is the step number
- **1/t Decay:** $\eta = \frac{\eta_0}{1+kt}$ where η_0 and k are hyperparameters and t is the step number



Tips for momentum

- The following schedule was suggested by Sutskever *et. al.*, 2013

$$\gamma_t = \min(1 - 2^{-1 - \log_2(\lfloor t/250 \rfloor + 1)}, \gamma_{\max})$$

where, γ_{\max} was chosen from $\{0.999, 0.995, 0.99, 0.9, 0\}$



Line Search



- 在实际应用中，通常使用 line search 来找到一个相对更好的学习率 η

- 在实际应用中，通常使用 line search 来找到一个相对更好的学习率 η

```
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    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)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
        w, b = best_w, best_b
```



- 在实际应用中，通常使用 line search 来找到一个相对更好的学习率 η
- 使用不同的学习率 η 来更新参数 w

```
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    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)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
    w, b = best_w, best_b
```



- 在实际应用中，通常使用 line search 来找到一个相对更好的学习率 η
- 使用不同的学习率 η 来更新参数 w
- 不同的 w ，计算不同的损失，选择产生最小损失的 w

```
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    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)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
    w, b = best_w, best_b
```

- 在实际应用中，通常使用 line search 来找到一个相对更好的学习率 η
- 使用不同的学习率 η 来更新参数 w
- 不同的 w ，计算不同的损失，选择产生最小损失的 w
- 每一步，从可能的学习率中选择一个最好的学习率 η

```
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    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)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
    w, b = best_w, best_b
```



- 在实际应用中，通常使用 line search 来找到一个相对更好的学习率 η
- 使用不同的学习率 η 来更新参数 w
- 不同的 w ，计算不同的损失，选择产生最小损失的 w
- 每一步，从可能的学习率中选择一个最好的学习率 η
- 缺点是啥？

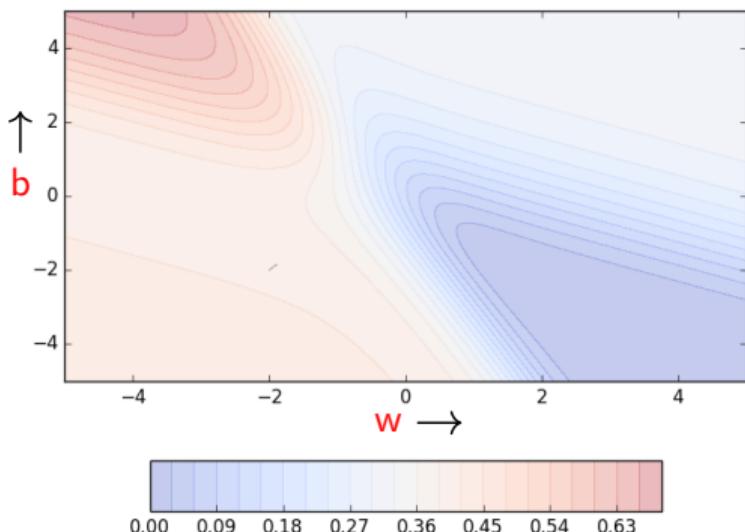
```
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    for i in range(max_epochs):
        dw, db = 0, 0
        for x,y in zip(X, Y):
            dw += grad_w(w, b, x, y)
            db += grad_b(w, b, x, y)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
    w, b = best_w, best_b
```

- 在实际应用中，通常使用 line search 来找到一个相对更好的学习率 η
- 使用不同的学习率 η 来更新参数 w
- 不同的 w , 计算不同的损失, 选择产生最小损失的 w
- 每一步, 从可能的学习率中选择一个最好的学习率 η
- 缺点是啥? 每一次更新, 带来更大的计算代价

```
def do_line_search_gradient_descent():
    w, b, etas = init_w, init_b, [0.1, 0.5, 1.0, 5.0, 10.0]
    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)
        min_error = 10000 #some large value
        best_w, best_b = w, b
        for eta in etas:
            tmp_w = w - eta * dw
            tmp_b = b - eta * db
            if error(tmp_w, tmp_b) < min_error:
                best_w = tmp_w
                best_b = tmp_b
                min_error = error(tmp_w, tmp_b)
        w, b = best_w, best_b
```

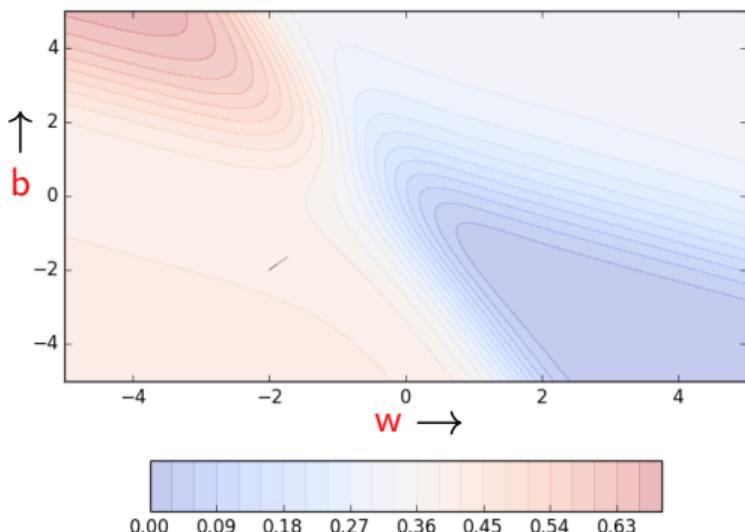


- 看一个实际的例子



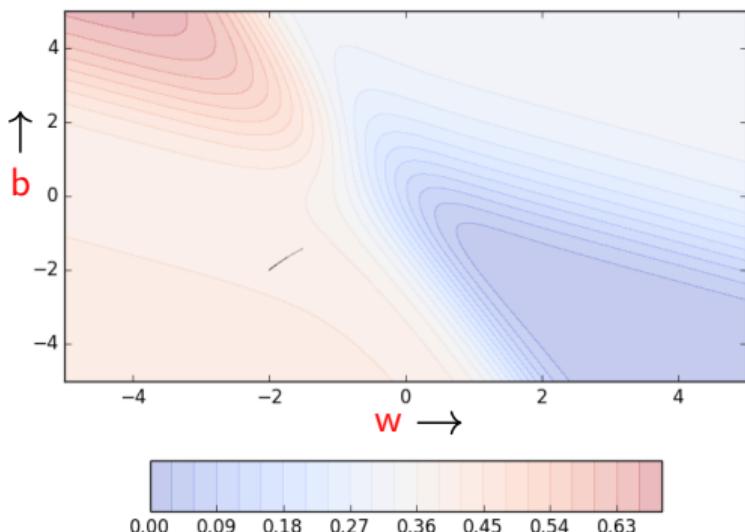


- 看一个实际的例子



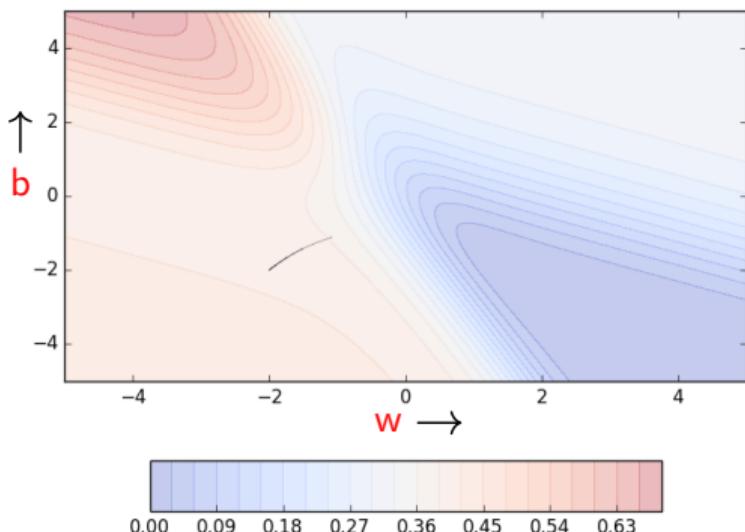


- 看一个实际的例子



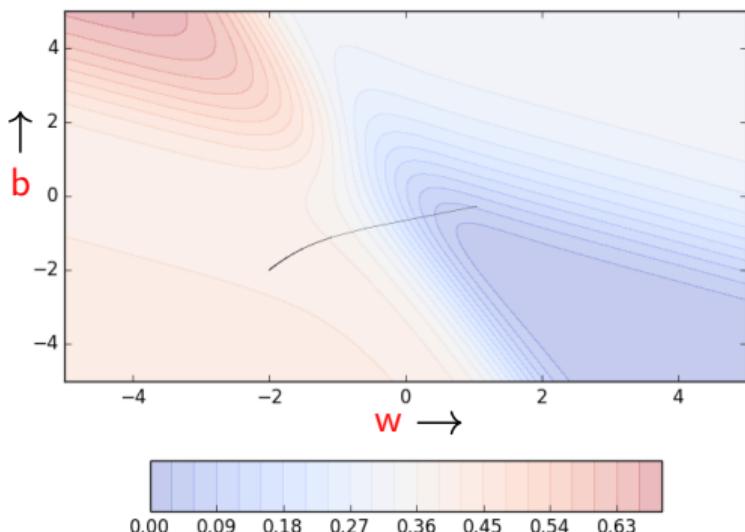


- 看一个实际的例子



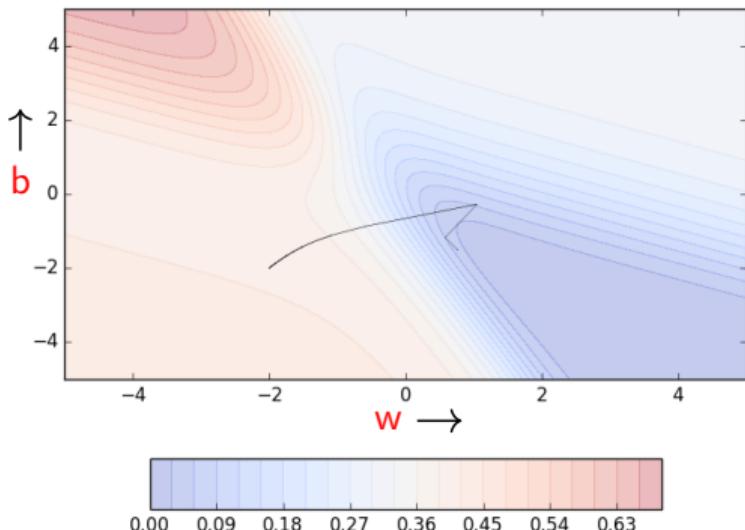


- 看一个实际的例子



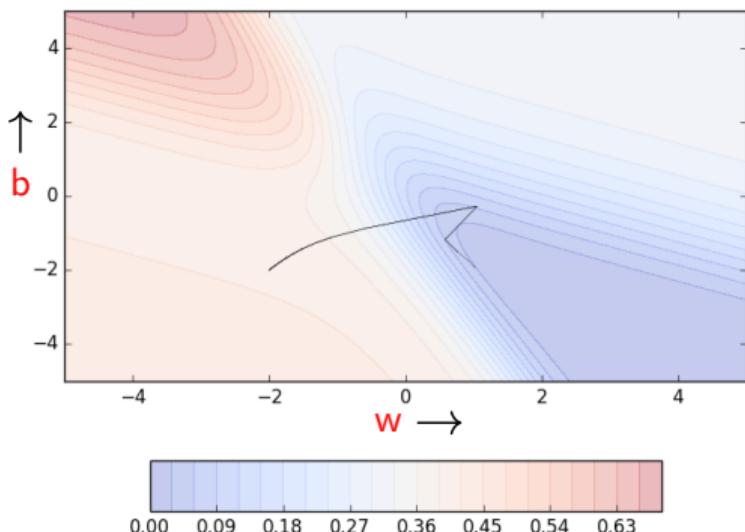


- 看一个实际的例子



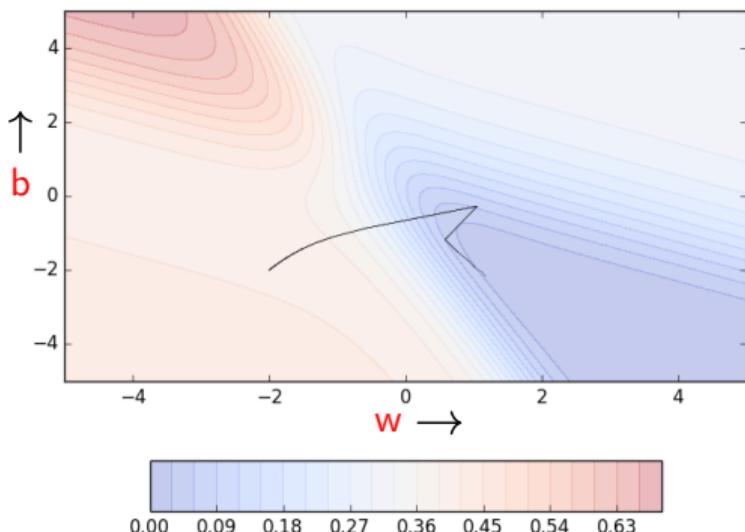


- 看一个实际的例子



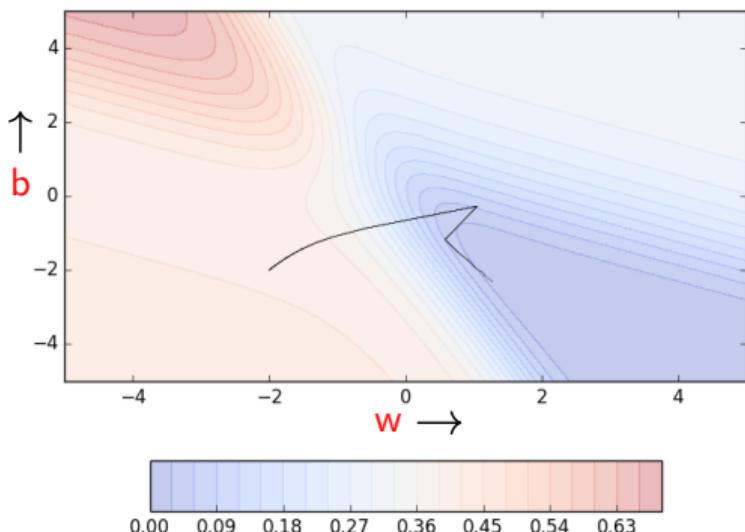


- 看一个实际的例子



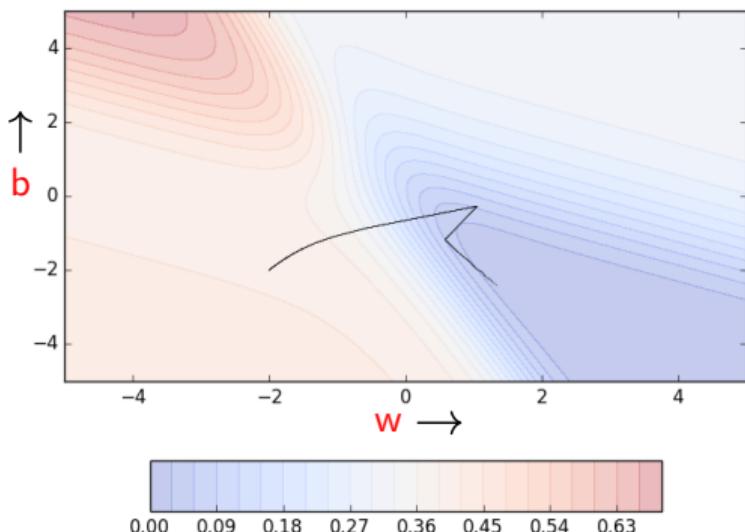


- 看一个实际的例子



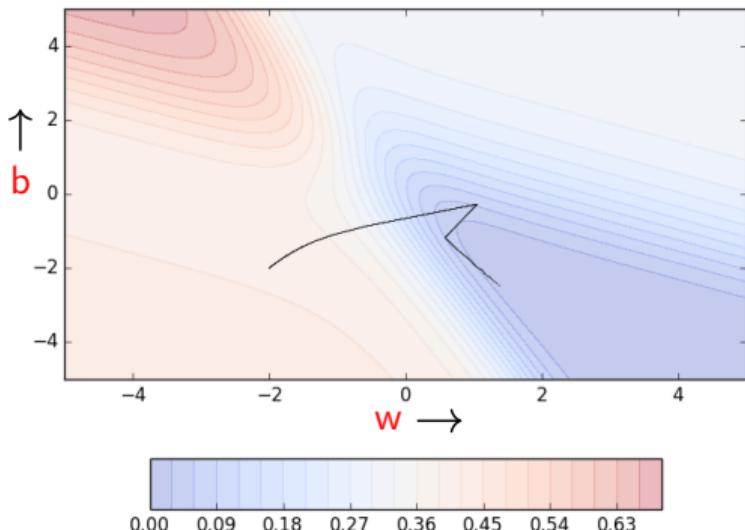


- 看一个实际的例子

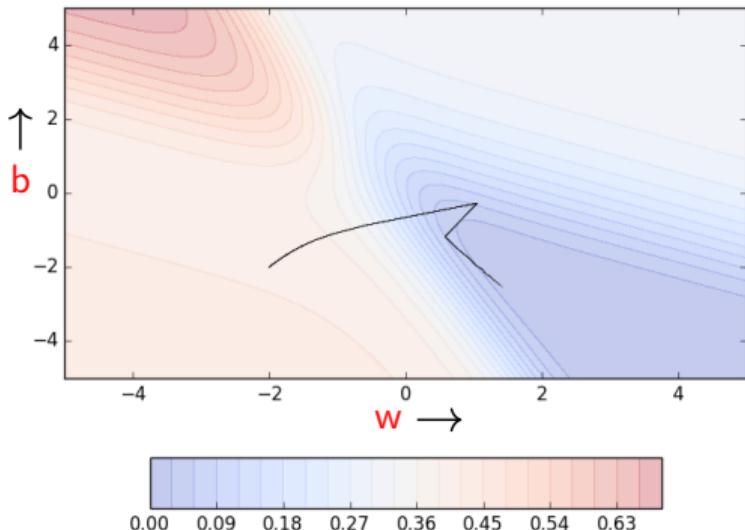




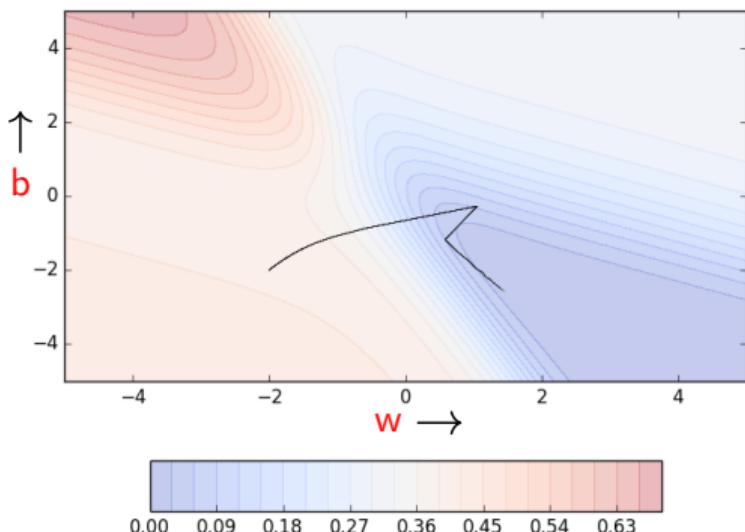
- 看一个实际的例子



- 看一个实际的例子

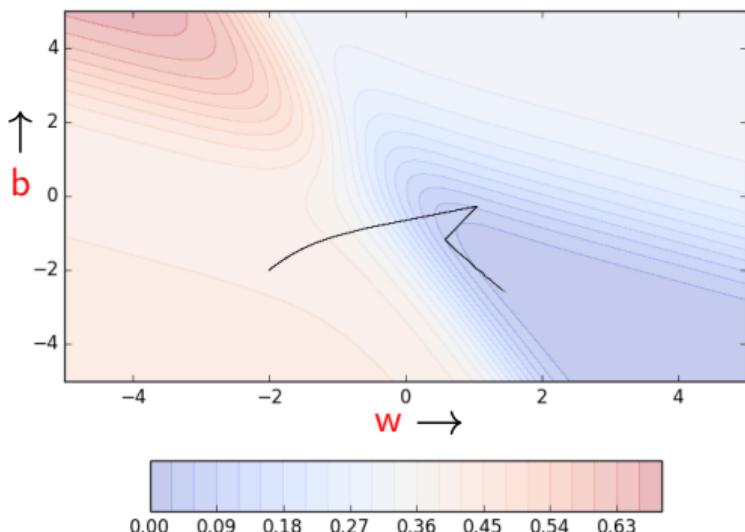


- 看一个实际的例子

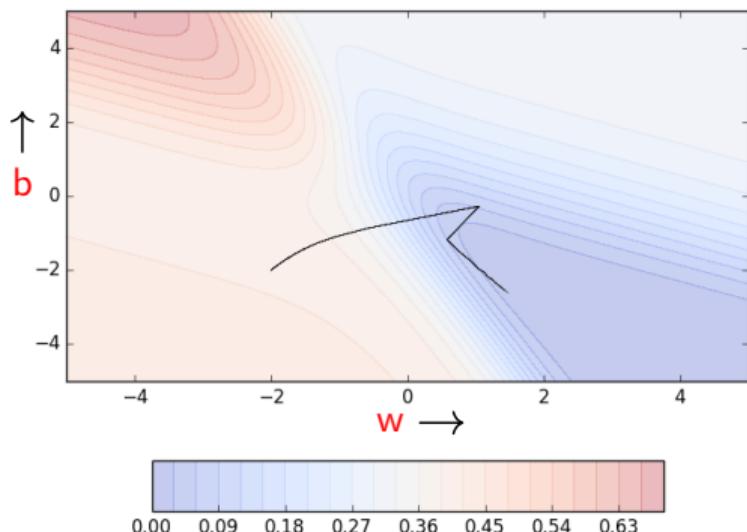




- 看一个实际的例子

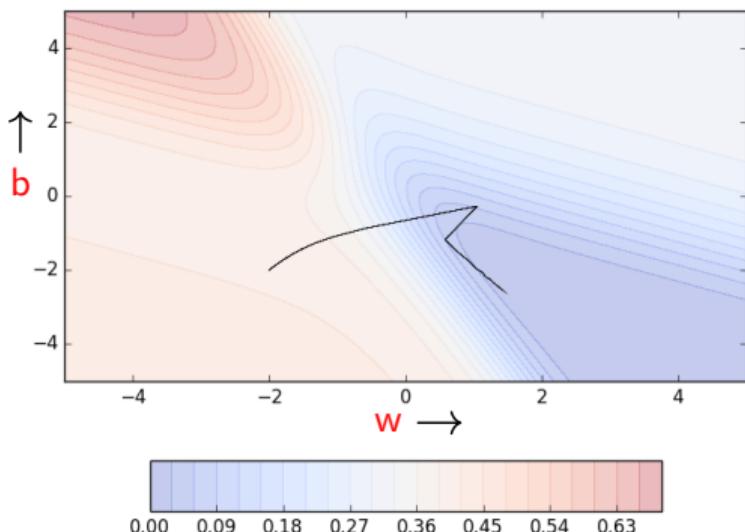


- 看一个实际的例子



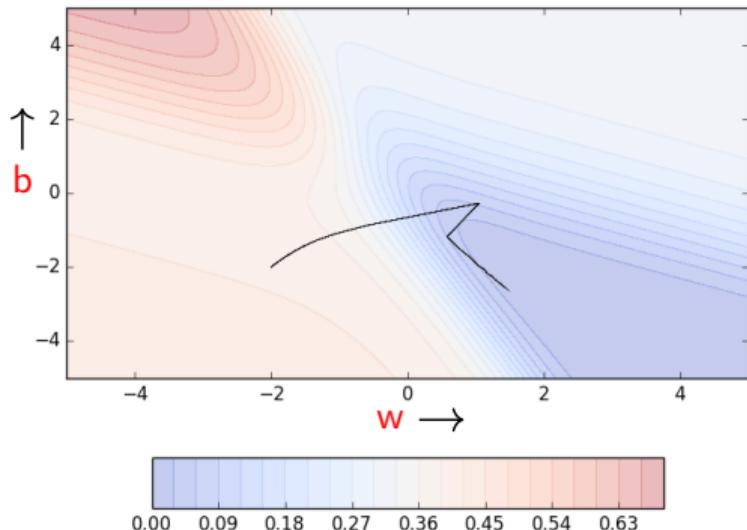


- 看一个实际的例子



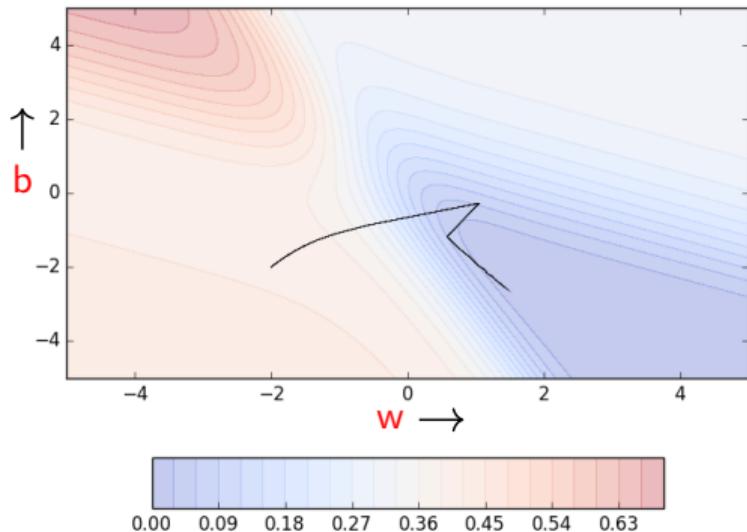


- 看一个实际的例子
- 比标准的梯度下降收敛的更快



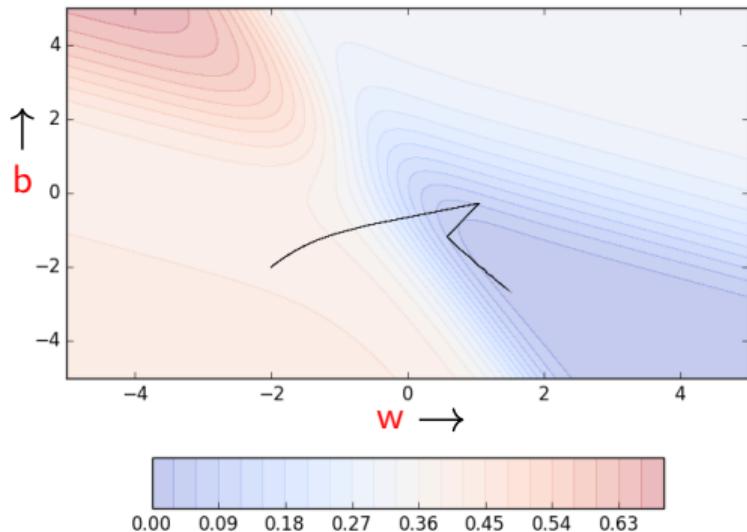


- 看一个实际的例子
- 比标准的梯度下降收敛的更快
- 也有一些振荡,





- 看一个实际的例子
- 比标准的梯度下降收敛的更快
- 也有一些振荡, 但与 momentum and NAG 产生的振荡不一样





- 看一个实际的例子
- 比标准的梯度下降收敛的更快
- 也有一些振荡, 但与 momentum and NAG 产生的振荡不一样

