

# Artificial Intelligence Foundation – JC3001

Lecture 40: Machine Learning – Regression III

**Prof. Aladdin Ayesh** (aladdin.ayesh@abdn.ac.uk)

**Dr. Binod Bhattacharai** (binod.bhattacharai@abdn.ac.uk)

**Dr. Gideon Ogunniye**, (g.ogunniye@abdn.ac.uk)

November 2025



UNIVERSITY OF  
ABERDEEN



Material adapted from:  
Russell and Norvig (AIMA Book): Chapter 19 (19.4–19.6)  
Sebastian Thrun (Stanford University / Udacity)  
Andrew Ng (Stanford University / Coursera)

# Course Progression

- Part 1: Introduction
  - ① Introduction to AI ✓
  - ② Agents ✓
- Part 2: Problem-solving
  - ① Search 1: Uninformed Search ✓
  - ② Search 2: Heuristic Search ✓
  - ③ Search 3: Local Search ✓
  - ④ Search 4: Adversarial Search ✓
- Part 3: Reasoning and Uncertainty
  - ① Reasoning 1: Constraint Satisfaction ✓
  - ② Reasoning 2: Logic and Inference ✓
  - ③ Probabilistic Reasoning 1: BNs ✓
  - ④ Probabilistic Reasoning 2: HMMs ✓
- Part 4: Planning
  - ① Planning 1: Intro and Formalism ✓
  - ② Planning 2: Algorithms & Heuristics ✓
  - ③ Planning 3: Hierarchical Planning ✓
  - ④ Planning 4: Stochastic Planning ✓
- Part 5: Learning
  - ① Learning 1: Intro to ML ✓
  - ② **Learning 2: Regression**
  - ③ Learning 3: Neural Networks
  - ④ Learning 4: Reinforcement Learning
- Part 6: Conclusion
  - ① Ethical Issues in AI
  - ② Conclusions and Discussion



# Outline

## 1 Gradient Descent

- ▶ Gradient Descent
- ▶ Regression for Classification

# Finding values for $w_0$ and $w_1$

## 1 Gradient Descent

Have some function  $Loss(w_0, w_1)$

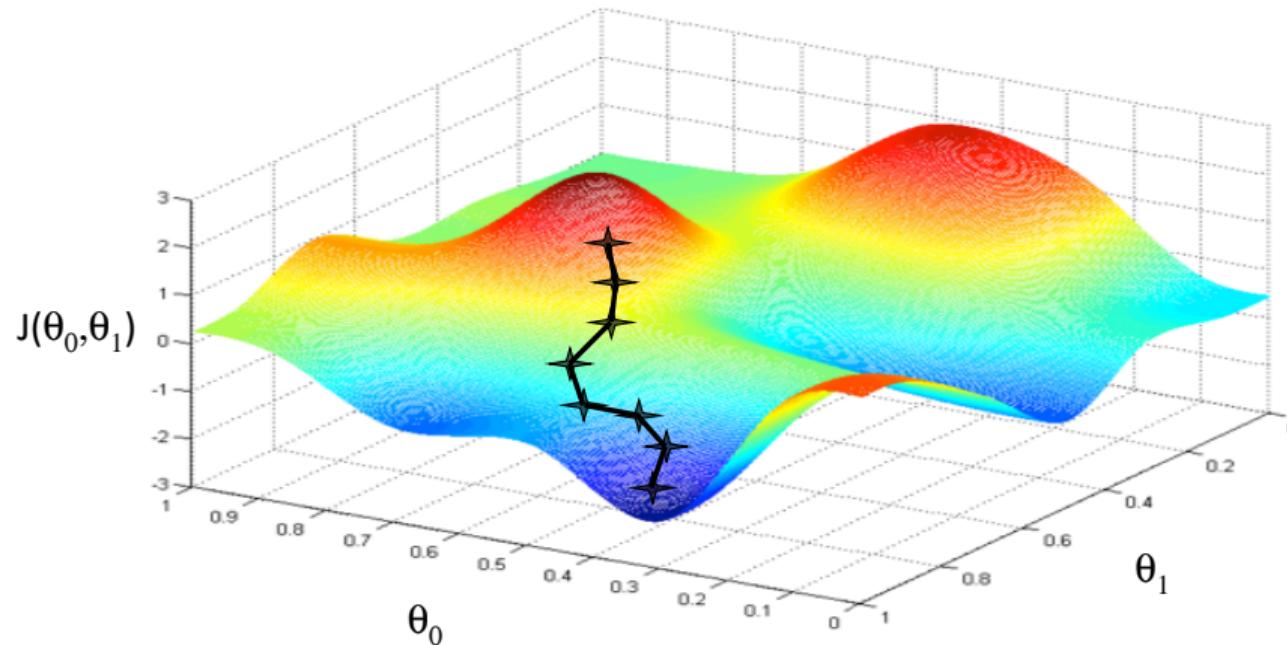
Want  $\min_{w_0, w_1} Loss(w_0, w_1)$

### Outline:

- Start with some  $w_0$  and  $w_1$
- Keep changing  $w_0$  and  $w_1$  to reduce  $Loss(w_0, w_1)$  until we hopefully end up at a minimum

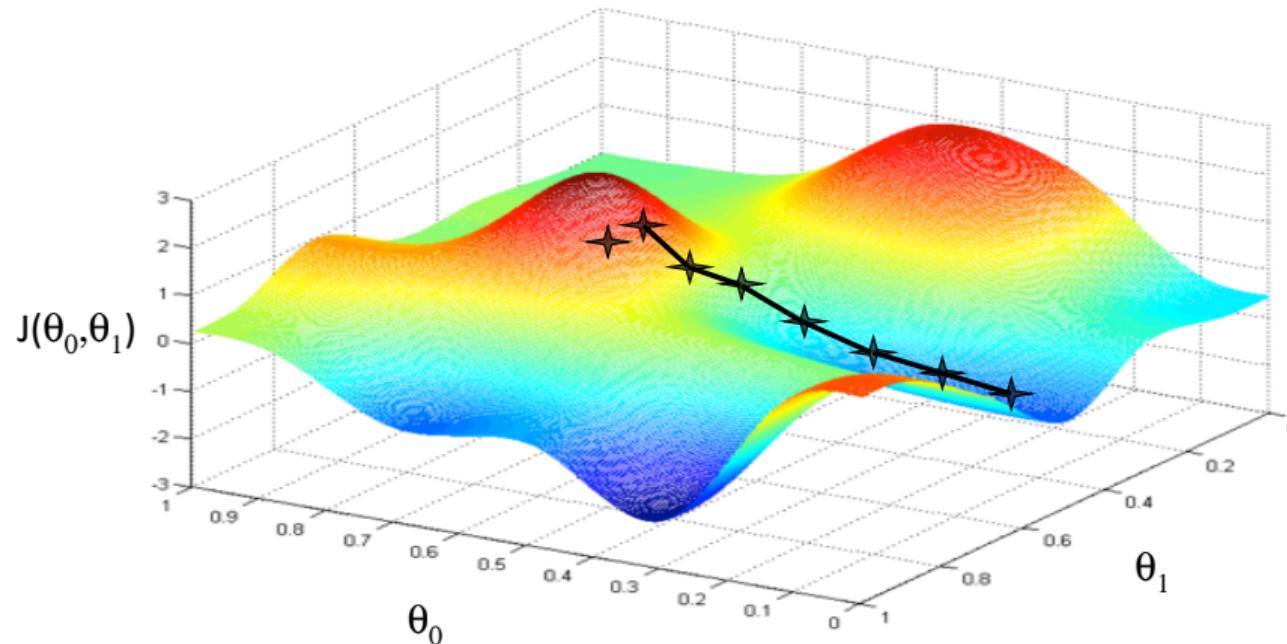
# Descending the gradient I

## 1 Gradient Descent



# Descending the gradient II

## 1 Gradient Descent



# Gradient Descent Algorithm

## 1 Gradient Descent

```
repeat until convergence {  
     $w_j := w_j - \alpha \frac{\partial}{\partial w_j} Loss(w_0, w_1)$       (for  $j = 0$  and  $j = 1$ )  
}
```

---

# Gradient Descent Algorithm

## 1 Gradient Descent

```
repeat until convergence {  
     $w_j := w_j - \alpha \frac{\partial}{\partial w_j} Loss(w_0, w_1)$       (for  $j = 0$  and  $j = 1$ )  
}
```

---

Correct: Simultaneous update

```
temp0 :=  $w_0 - \alpha \frac{\partial}{\partial w_0} Loss(w_0, w_1)$   
temp1 :=  $w_1 - \alpha \frac{\partial}{\partial w_1} Loss(w_0, w_1)$   
 $w_0 := temp0$   
 $w_1 := temp1$ 
```

# Gradient Descent Algorithm

## 1 Gradient Descent

```

repeat until convergence {
     $w_j := w_j - \alpha \frac{\partial}{\partial w_j} Loss(w_0, w_1)$       (for  $j = 0$  and  $j = 1$ )
}

```

---

Correct: Simultaneous update

```

temp0 :=  $w_0 - \alpha \frac{\partial}{\partial w_0} Loss(w_0, w_1)$ 
temp1 :=  $w_1 - \alpha \frac{\partial}{\partial w_1} Loss(w_0, w_1)$ 
 $w_0 := temp0$ 
 $w_1 := temp1$ 

```

Incorrect

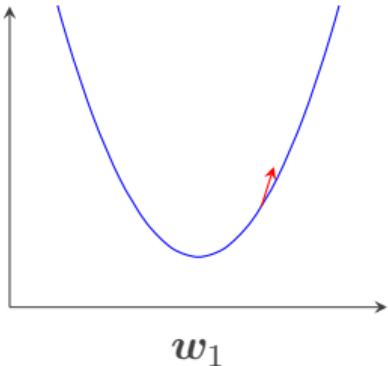
```

temp0 :=  $w_0 - \alpha \frac{\partial}{\partial w_0} Loss(w_0, w_1)$ 
 $w_0 := temp0$ 
temp1 :=  $w_1 - \alpha \frac{\partial}{\partial w_1} Loss(w_0, w_1)$ 
 $w_1 := temp1$ 

```

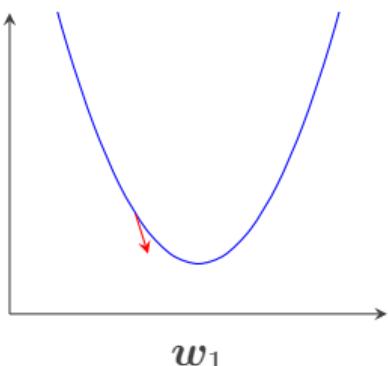
# Gradient Descent—Derivative Behaviour

## 1 Gradient Descent



$$w_1 - \alpha \underbrace{\frac{d}{dw_1} Loss(w_1)}_{\geq 0}$$

$w_1 - \alpha(\text{positive number})$



$$w_1 - \alpha \underbrace{\frac{d}{dw_1} Loss(w_1)}_{\leq 0}$$

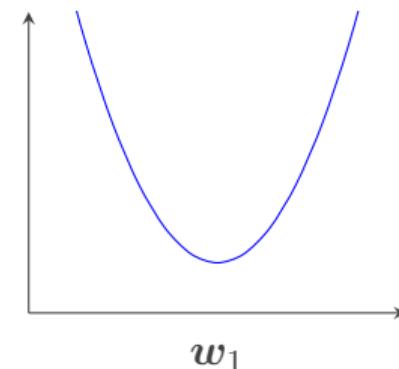
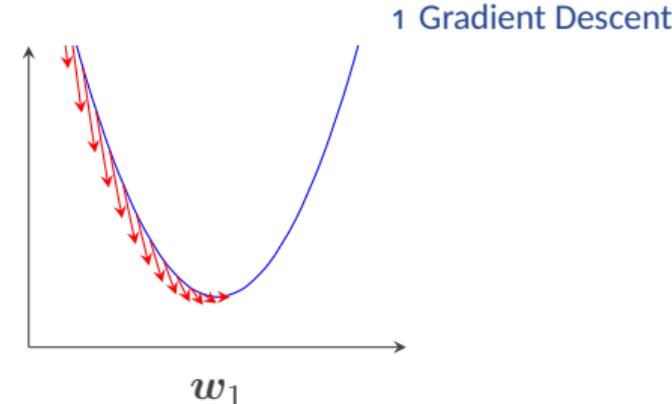
$w_1 - \alpha(\text{negative number})$

## Gradient Descent – Learning Rate

$$w_1 := w_1 - \alpha \frac{\partial}{\partial w_1} Loss(w_1)$$

If  $\alpha$  is too small, gradient descent can be slow

If  $\alpha$  is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



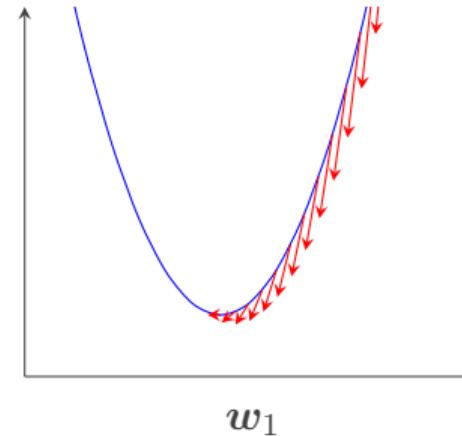
# Gradient Descent – Fixed Alpha

## 1 Gradient Descent

Gradient descent can converge to a local minimum, even with the learning rate  $\alpha$  fixed.

$$\mathbf{w}_1 := \mathbf{w}_1 - \alpha \frac{\partial}{\partial \mathbf{w}_1} Loss(\mathbf{w}_1)$$

As we approach a local minimum, gradient descent will automatically take smaller steps. So, no need to decrease  $\alpha$  over time.



# Derivative of Loss( $\vec{w}$ )

## 1 Gradient Descent

$$\begin{aligned}\frac{\partial}{\partial \mathbf{w}_1} \text{Loss}(\mathbf{w}_1) &= \frac{\partial}{\partial \mathbf{w}_1} \frac{1}{2m} \sum_{i=1}^m (h_{\mathbf{w}}(x^{(i)}) - y^{(i)})^2 \\ &= \frac{\partial}{\partial \mathbf{w}_1} \frac{1}{2m} \sum_{i=1}^m (\mathbf{w}_0 + \mathbf{w}_1 x^{(i)} - y^{(i)})^2\end{aligned}$$

$$\begin{aligned}j = 0 : \frac{\partial}{\partial \mathbf{w}_0} \text{Loss}(\mathbf{w}_0) &= \frac{1}{m} \sum_{i=1}^m (h_{\mathbf{w}}(x^{(i)}) - y^{(i)}) \\ j = 1 : \frac{\partial}{\partial \mathbf{w}_1} \text{Loss}(\mathbf{w}_1) &= \frac{1}{m} \sum_{i=1}^m (h_{\mathbf{w}}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}\end{aligned}$$

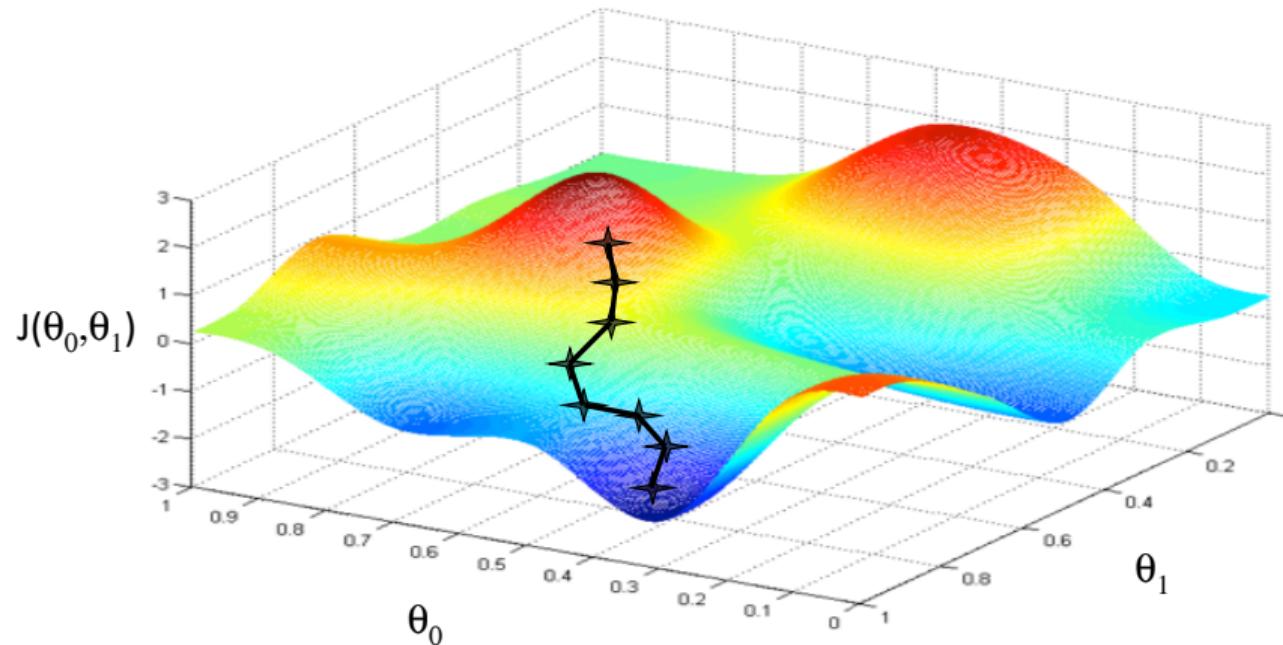
# Gradient Descent Algorithm

## 1 Gradient Descent

```
repeat until convergence {  
     $w_0 := w_0 - \alpha \frac{1}{m} \sum_{i=1}^m (h_w(x^{(i)}) - y^{(i)})$   
     $w_1 := w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (h_w(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$   
}
```

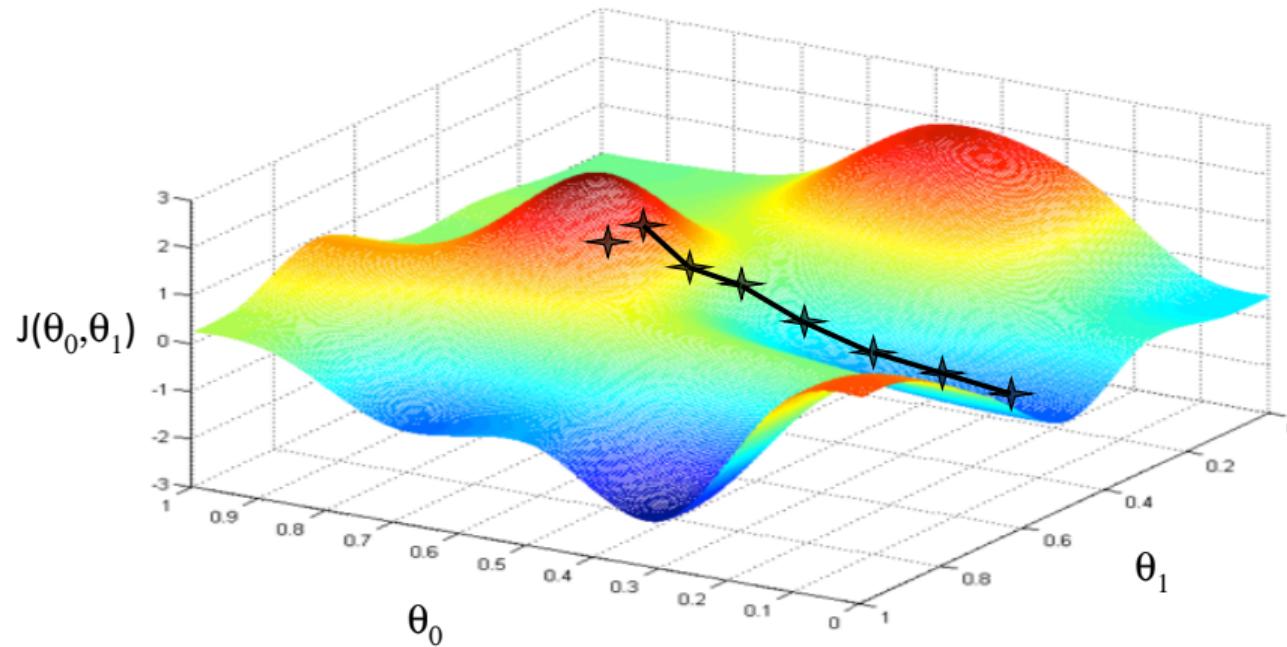
# Descending the gradient I

## 1 Gradient Descent



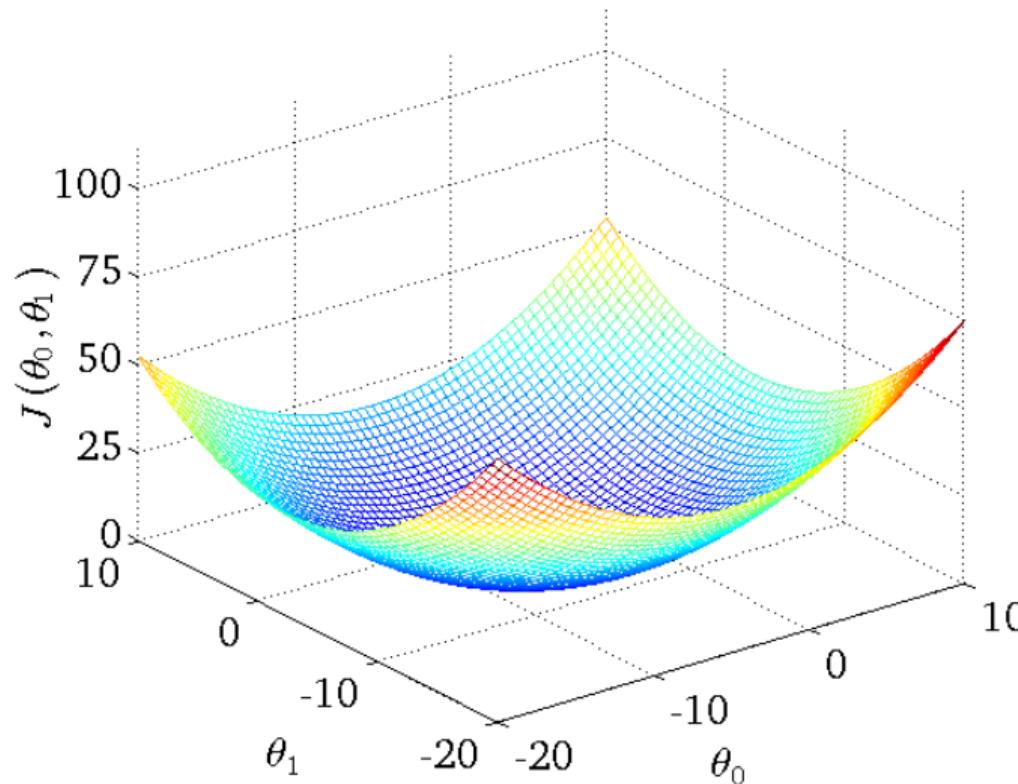
# Descending the gradient II

## 1 Gradient Descent



# Convex Functions

## 1 Gradient Descent

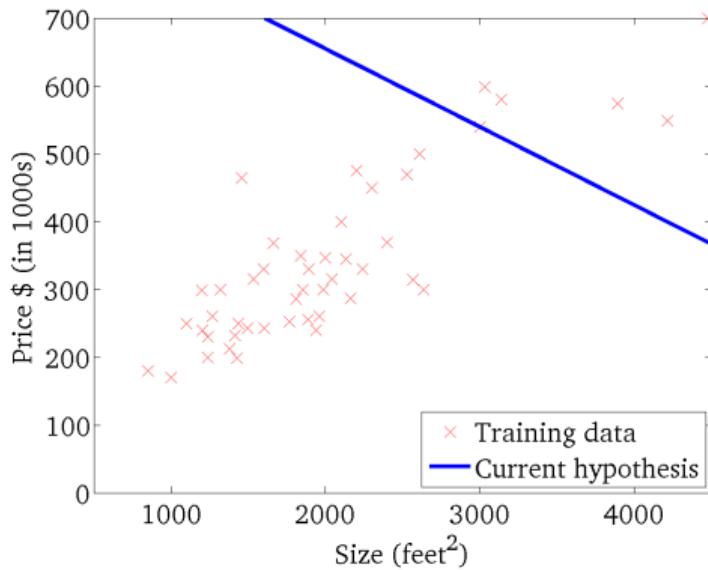


# Gradient Descent in Action 1

## 1 Gradient Descent

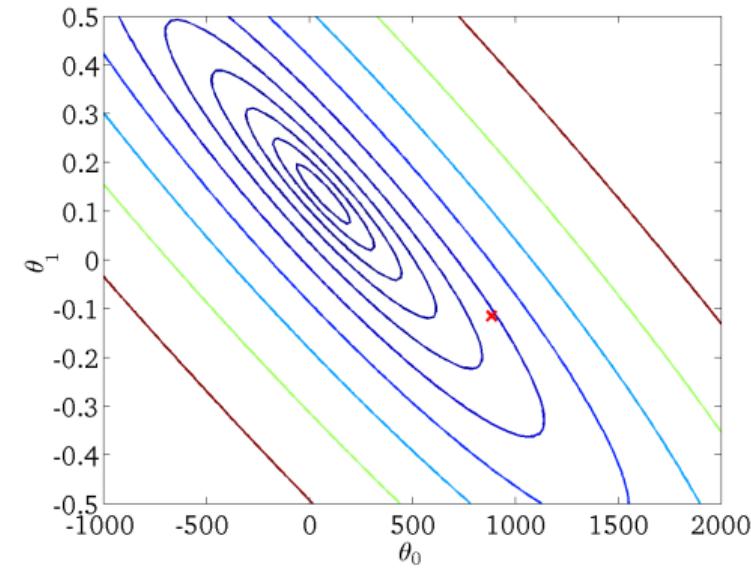
$$h_w(x)$$

(for fixed  $w_0, w_1$ , this is a function of  $x$ )



$$Loss(w_0, w_1)$$

(function of the parameters  $w_0, w_1$ )

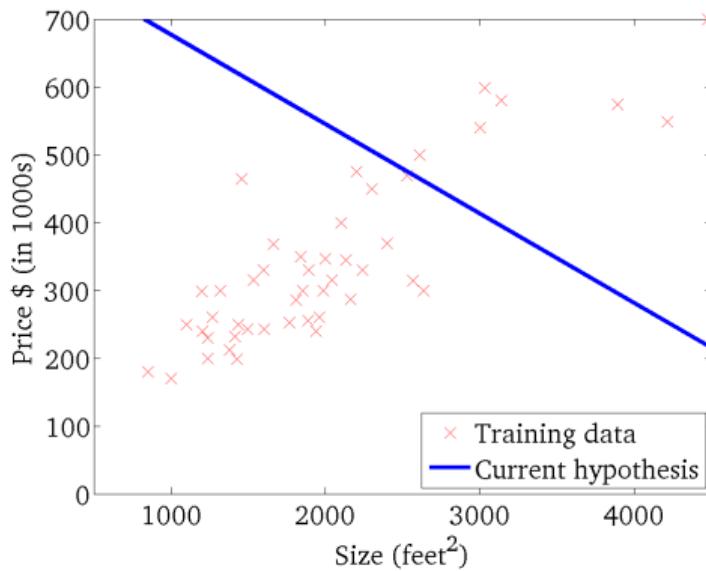


## Gradient Descent in Action 2

### 1 Gradient Descent

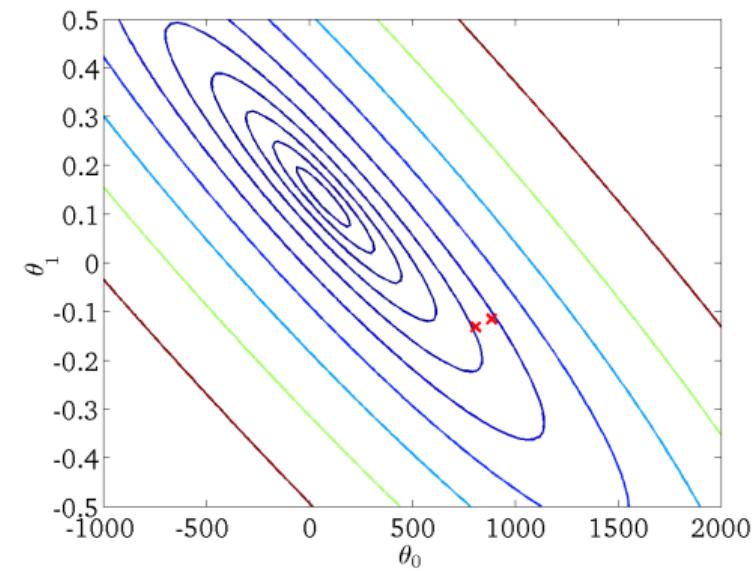
$$h_{\mathbf{w}}(x)$$

(for fixed  $w_0, w_1$ , this is a function of  $x$ )



$$Loss(\mathbf{w}_0, \mathbf{w}_1)$$

(function of the parameters  $w_0, w_1$ )

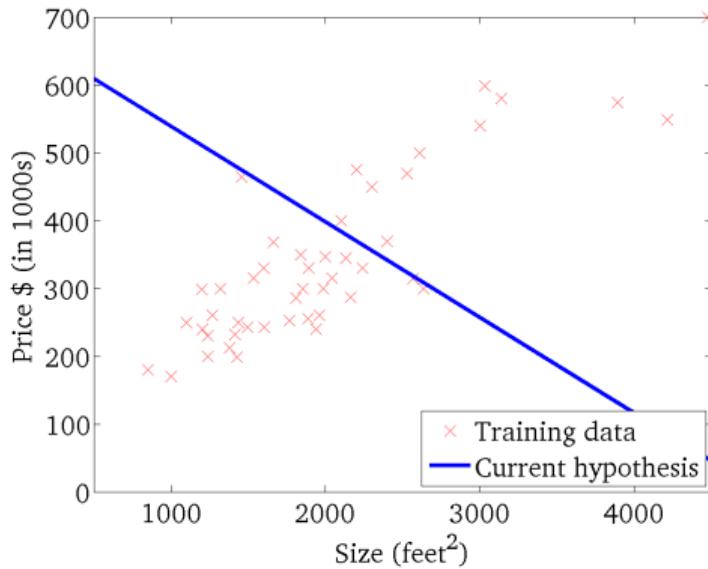


# Gradient Descent in Action 3

## 1 Gradient Descent

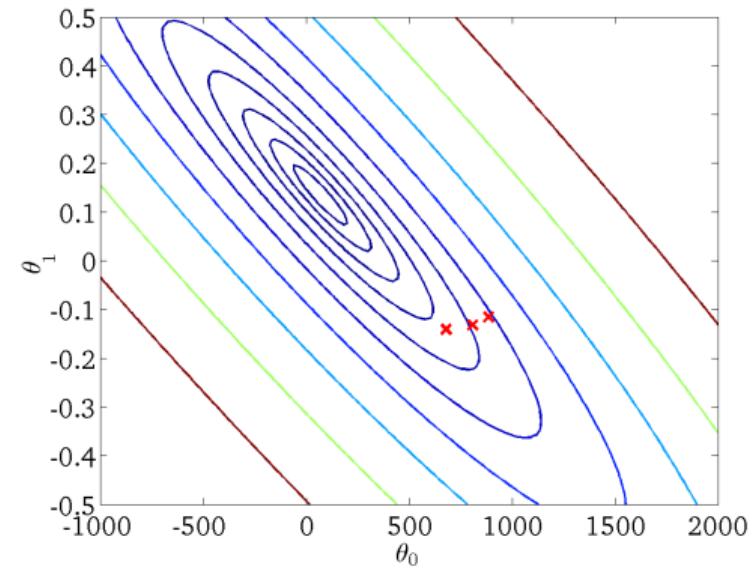
$$h_w(x)$$

(for fixed  $w_0, w_1$ , this is a function of  $x$ )



$$Loss(w_0, w_1)$$

(function of the parameters  $w_0, w_1$ )

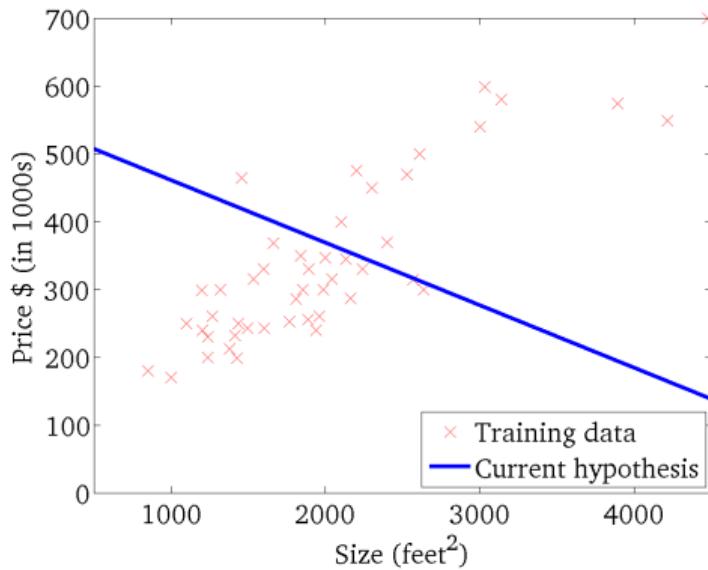


# Gradient Descent in Action 4

## 1 Gradient Descent

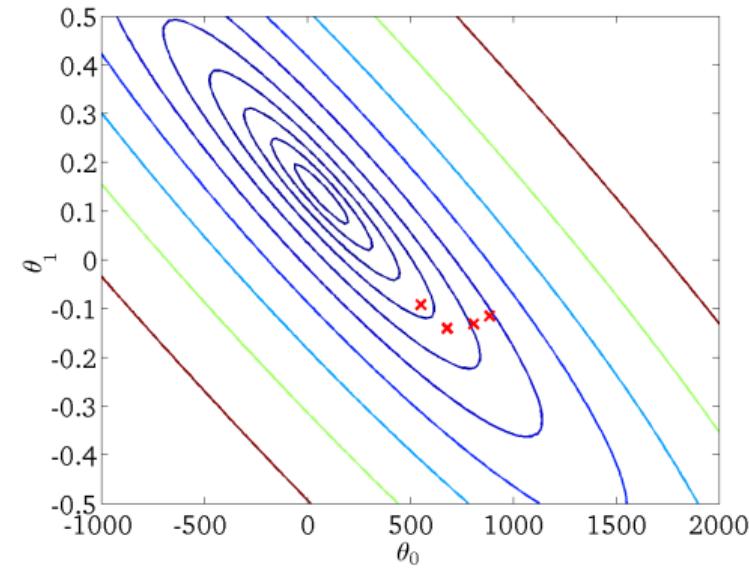
$$h_w(x)$$

(for fixed  $w_0, w_1$ , this is a function of  $x$ )



$$Loss(w_0, w_1)$$

(function of the parameters  $w_0, w_1$ )

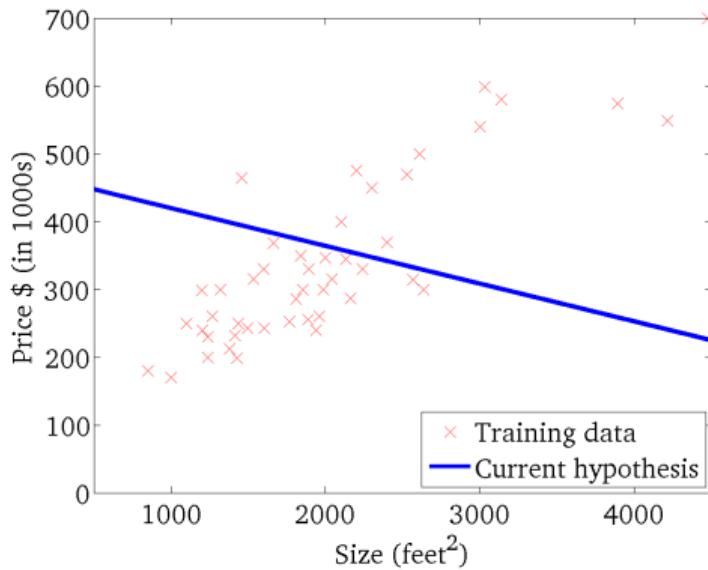


# Gradient Descent in Action 5

## 1 Gradient Descent

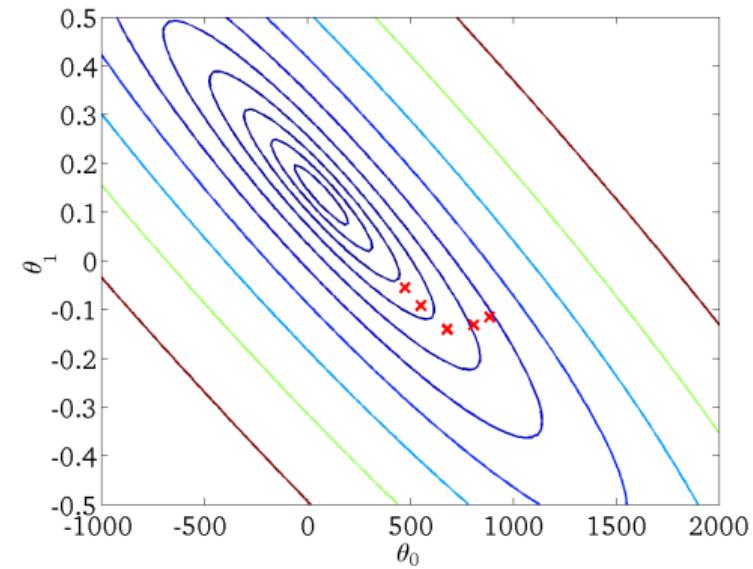
$$h_w(x)$$

(for fixed  $w_0, w_1$ , this is a function of  $x$ )



$$Loss(w_0, w_1)$$

(function of the parameters  $w_0, w_1$ )

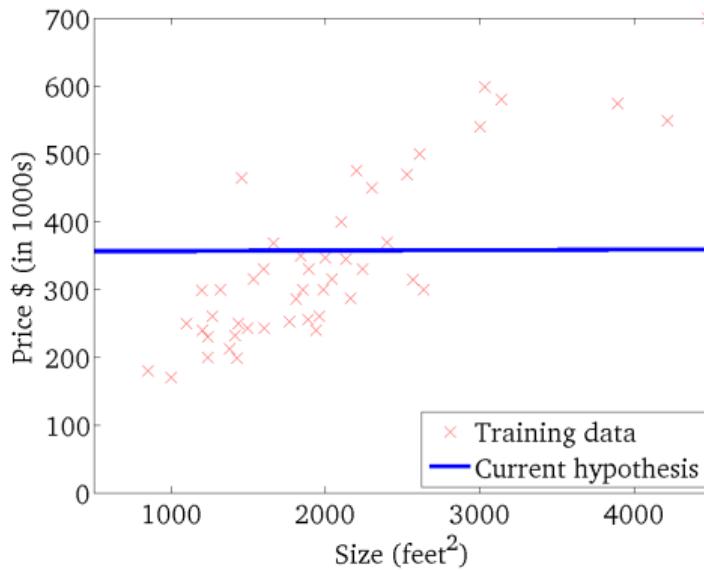


## Gradient Descent in Action 6

### 1 Gradient Descent

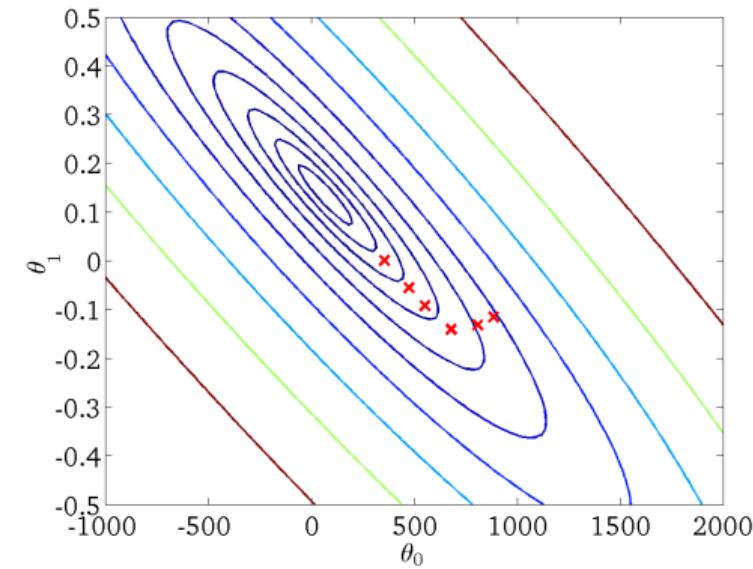
$$h_{\mathbf{w}}(x)$$

(for fixed  $w_0, w_1$ , this is a function of  $x$ )



$$\text{Loss}(\mathbf{w}_0, \mathbf{w}_1)$$

(function of the parameters  $w_0, w_1$ )

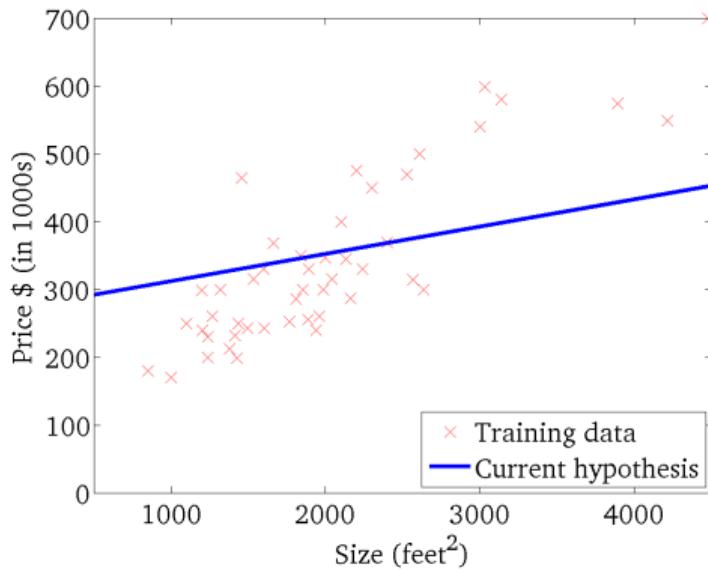


# Gradient Descent in Action 7

## 1 Gradient Descent

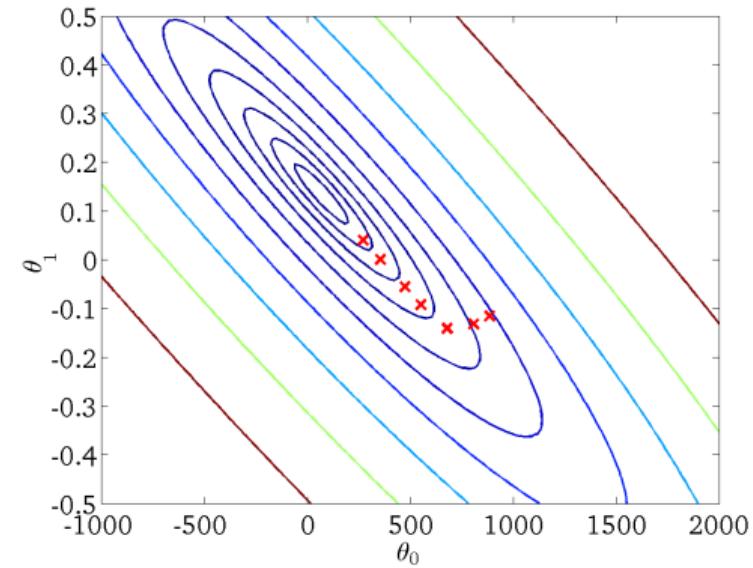
$$h_w(x)$$

(for fixed  $w_0, w_1$ , this is a function of  $x$ )



$$Loss(w_0, w_1)$$

(function of the parameters  $w_0, w_1$ )

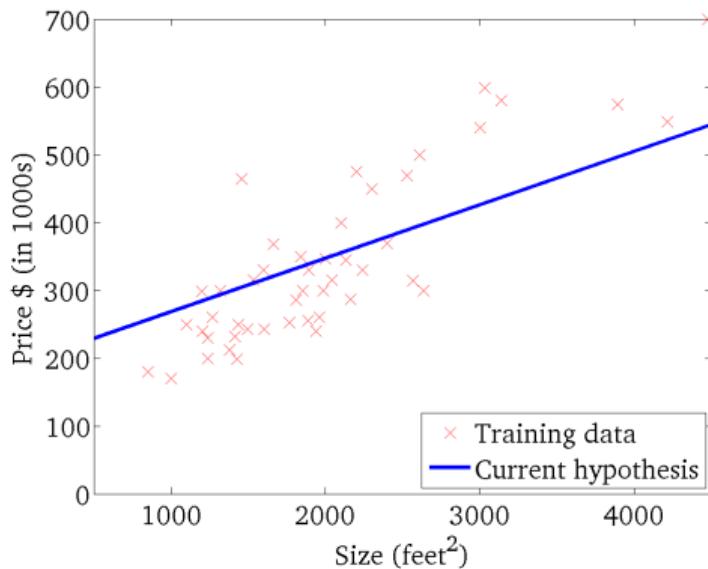


## Gradient Descent in Action 8

### 1 Gradient Descent

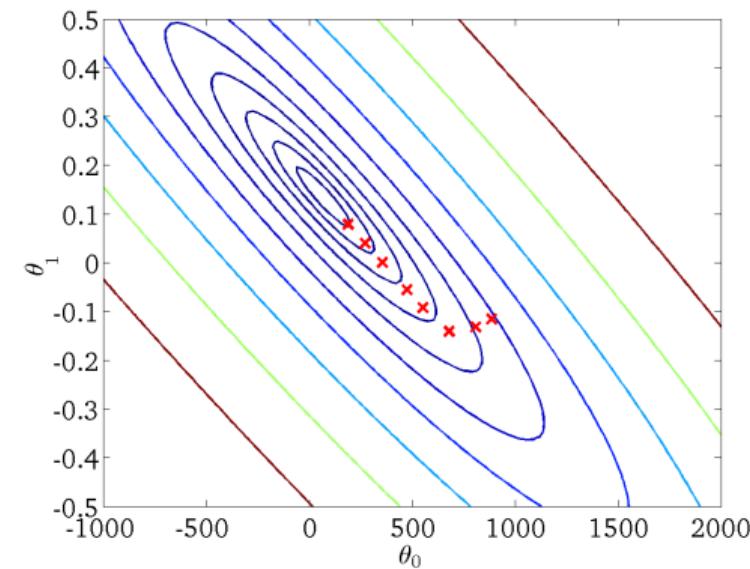
$$h_{\mathbf{w}}(x)$$

(for fixed  $w_0, w_1$ , this is a function of  $x$ )



$$Loss(\mathbf{w}_0, \mathbf{w}_1)$$

(function of the parameters  $w_0, w_1$ )

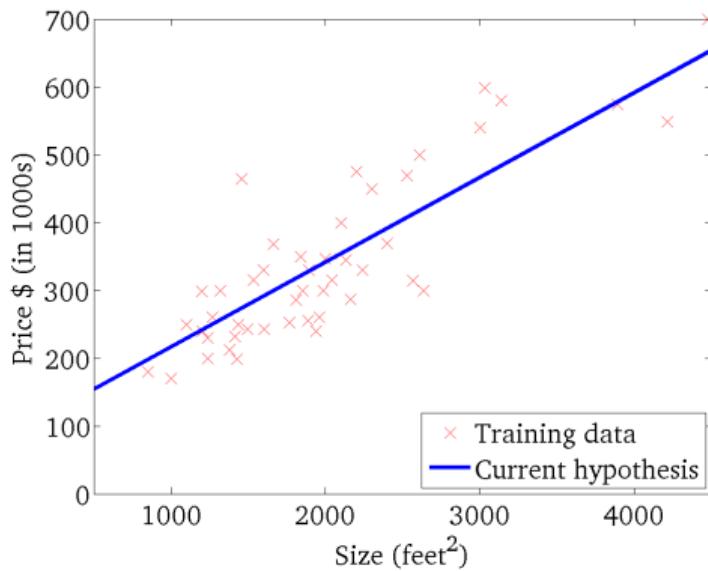


# Gradient Descent in Action 9

## 1 Gradient Descent

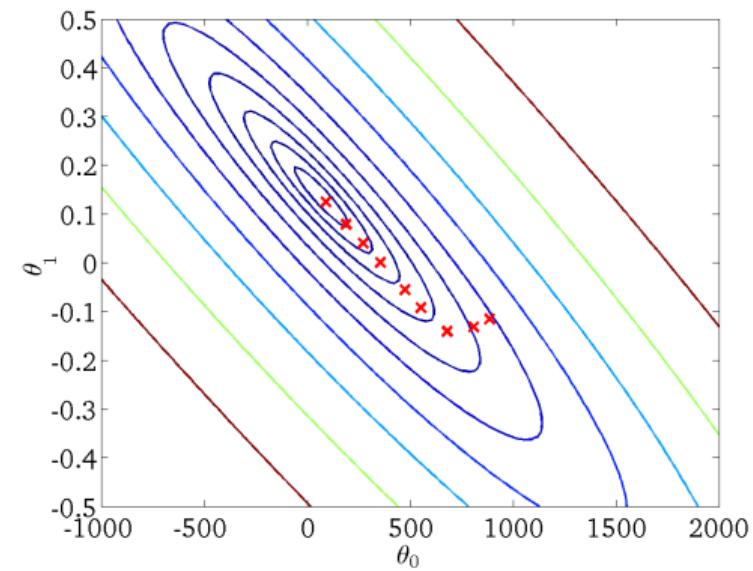
$$h_w(x)$$

(for fixed  $w_0, w_1$ , this is a function of  $x$ )



$$Loss(w_0, w_1)$$

(function of the parameters  $w_0, w_1$ )





# Outline

## 2 Regression for Classification

- ▶ Gradient Descent
- ▶ Regression for Classification

# Linear classifiers with a hard threshold

## 2 Regression for Classification

- Linear functions can be used to do classification as well as regression.
- **Decision boundary:** a line (or a surface, in higher dimensions) that separates the two classes.
- **Linear separator:** linear decision boundary for linearly separable data
- $h$ : result of passing the linear function  $\mathbf{w} \cdot \mathbf{x}$  through a threshold function:

$$h_{\mathbf{w}}(\mathbf{x}) = \text{Threshold}(\mathbf{w} \cdot \mathbf{x})$$

where

$$\text{Threshold}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

# Linear classifiers with a hard threshold

## 2 Regression for Classification

- $h$ : result of passing the linear function  $\mathbf{w} \cdot \mathbf{x}$  through a threshold function:

$$h_{\mathbf{w}}(\mathbf{x}) = \text{Threshold}(\mathbf{w} \cdot \mathbf{x})$$

where

$$\text{Threshold}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ 0 & \text{otherwise} \end{cases}$$

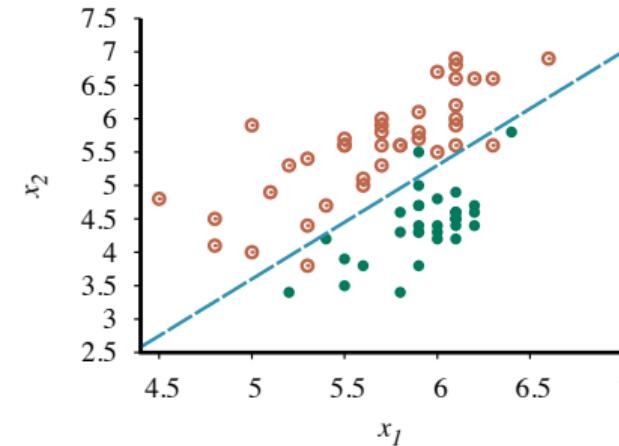
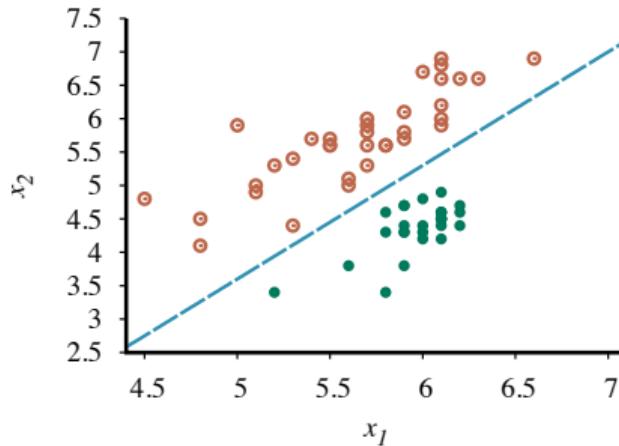
- Update rule is identical as for linear regression (**perceptron update rule**):

$$\mathbf{w}_i \leftarrow \mathbf{w}_i + \alpha (y - h_{\mathbf{w}}(\mathbf{x})) \times x_i$$

- Possibilities of outputs for weight update during training:
  - Correct output: weight unchanged
  - $y$  is 1 but  $h_{\mathbf{w}}(\mathbf{x})$  is 0:  $\mathbf{w}_i$  is increased when the corresponding input  $x_i$  is positive and decreased when  $x_i$  is negative.
  - $y$  is 0 but  $h_{\mathbf{w}}(\mathbf{x})$  is 1:  $\mathbf{w}_i$  is decreased when the corresponding input  $x_i$  is positive and increased when  $x_i$  is negative.

# Linear classifiers with a hard threshold

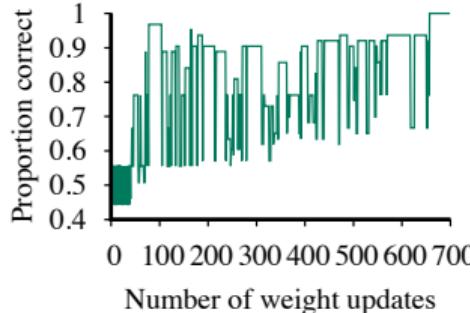
## 2 Regression for Classification



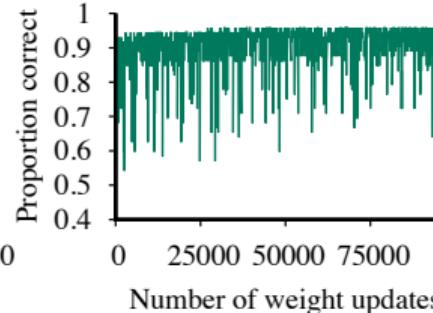
- (Left) Plot of two seismic data parameters, body wave magnitude  $x_1$  and surface wave magnitude  $x_2$ , for earthquakes (open orange circles) and nuclear explosions (green circles) occurring between 1982 and 1990 in Asia and the Middle East. Also shown is a decision boundary between the classes.
- The same domain with more data points. Earthquakes and explosions are no longer linearly separable.

# Linear classifiers with a hard threshold

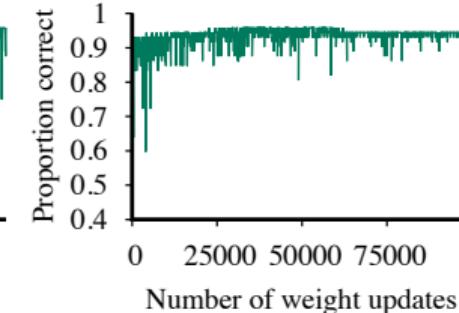
## 2 Regression for Classification



(a)



(b)



(c)

- (a) Plot of total training-set accuracy vs. number of iterations through the training set for the perceptron learning rule, given the earthquake/explosion data
- (b) The same plot for the noisy, nonseparable earthquake/explosion note the change in scale of the x-axis.
- (c) The same plot as in (b), with a learning rate schedule  $\alpha(t) = 1000/(1000 + t)$ .

# Linear classification with logistic regression

## 2 Regression for Classification

- Softening the threshold function— approximating the hard threshold with a continuous, differentiable function
- Logistic function:

$$\text{Logistic}(z) = \frac{1}{1 + e^{-z}}$$

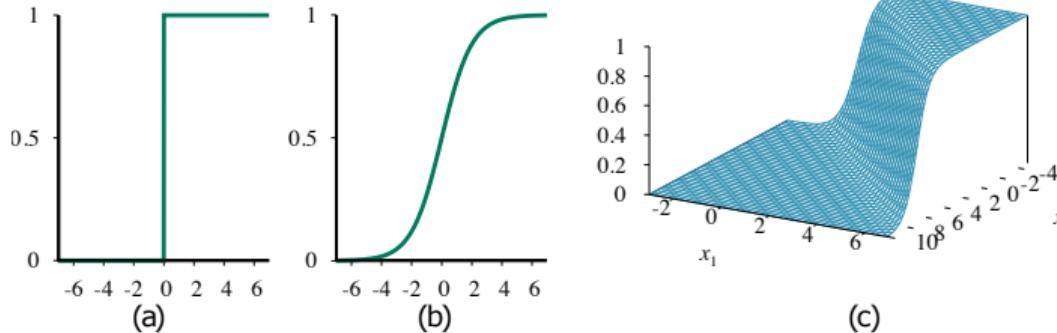
- Used to replace the threshold function:

$$h_w(x) = \text{Logistic}(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}}$$

- **Logistic regression:** process of fitting the weights of this model to minimize loss on a data set

# Linear classification with logistic regression

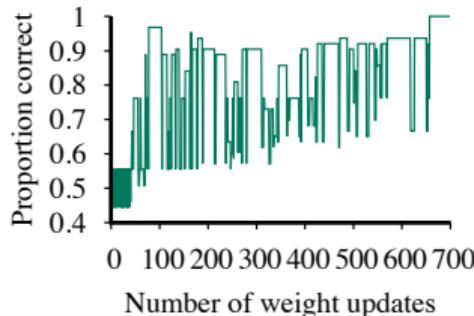
## 2 Regression for Classification



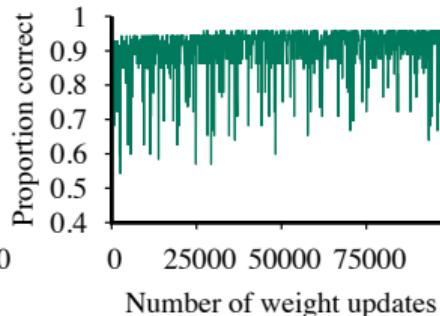
- (a) The hard threshold function  $\text{Threshold}(z)$  with 0/1 output.  
 Note that the function is nondifferentiable at  $z = 0$ .
- (b) The logistic function (sigmoid function),  $\text{Logistic}(z) = \frac{1}{1+e^{-z}}$
- (c) Plot of a logistic regression hypothesis  $h_w(x) = \text{Logistic}(w \cdot x)$

# Linear classification with logistic regression

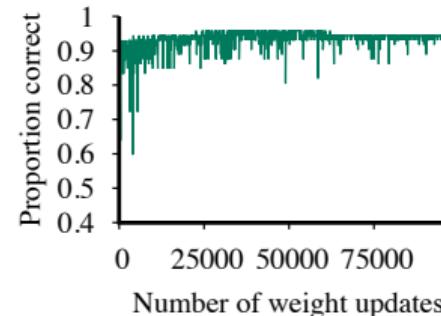
## 2 Regression for Classification



(a)



(b)

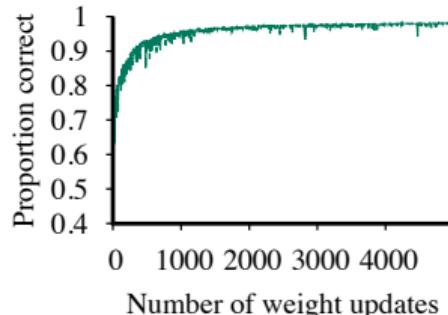


(c)

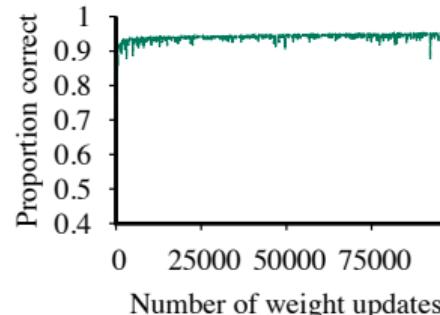
- Logistic regression on previous data. The plot in (a) covers 5000 iterations rather than 700, while the plots in (b) and (c) use the same scale as before.

# Linear classification with logistic regression

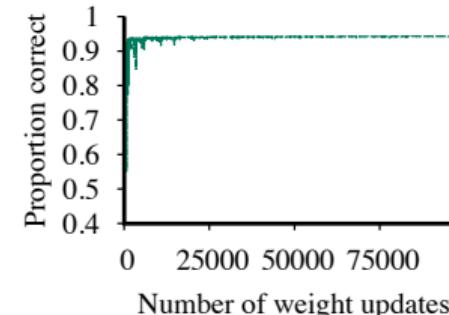
## 2 Regression for Classification



(a)



(b)



(c)

- Logistic regression on previous data. The plot in (a) covers 5000 iterations rather than 700, while the plots in (b) and (c) use the same scale as before.

# Regression Summary

## 2 Regression for Classification

- Linear Regression
- Gradient Descent
- Linear Classifiers
  - Thresholds for Classification
  - Logistic Regression



UNIVERSITY OF  
ABERDEEN



# Any Questions.