CPSC 4420/6420

Artificial Intelligence

23 – Linear Regression, Gradient Descent

November 10, 2020

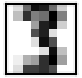
Announcements

- Project 4 is due on 11/12
- Quiz 7 is due on 11/12

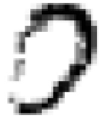
Supervised Learning

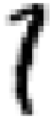
Example: Digit classification

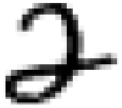
- Input: images / pixel grids
  - Each input maps to a feature vector
  - E.g. one feature (variable) for each grid position based on pixel intensity

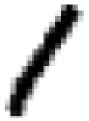
 $\rightarrow \langle 0, 0, 1, 1, 0 \ldots 0 \rangle$

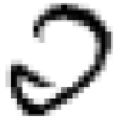
- Output: a digit 0-9
- Setup
  - Get a large collection of example images, each labeled with a digit
  - Note: someone has to hand label all this data!
  - Want to learn to predict labels of new, future digit images


0


1


2


1


??

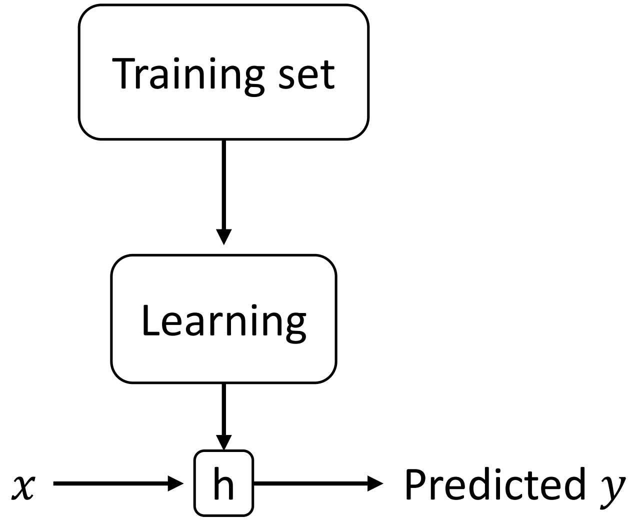
Example: House pricing

- Input: houses in a neighborhood
  - Each input maps to a feature vector
  - E.g. the size of the house, the number of bedrooms, etc.
- Output: house price
- Setup
  - Collect a large number of training examples
  - Want to learn to predict the prices of other houses in the neighborhood

Living area ($feet^2$)	Price ($1000s)
2100	410
1650	320
2200	387
1432	214
…	…

Supervised learning

- Let x denote the "input" variables/features, and y the "output" or target variable
- Let $\mathcal{X}$ denote the space of input values, and $\mathcal{Y}$ the space of output values
- Supervised learning
  - We are given a training set, consisting of a list of m training examples $\{(x^{(i)}, y^{(i)}), i = 1 \ldots m\}$
  - Our goal is to learn a function $h : \mathcal{X} \mapsto \mathcal{Y}$, so that $y^{(i)} \approx h(x^{(i)})$ for each training example



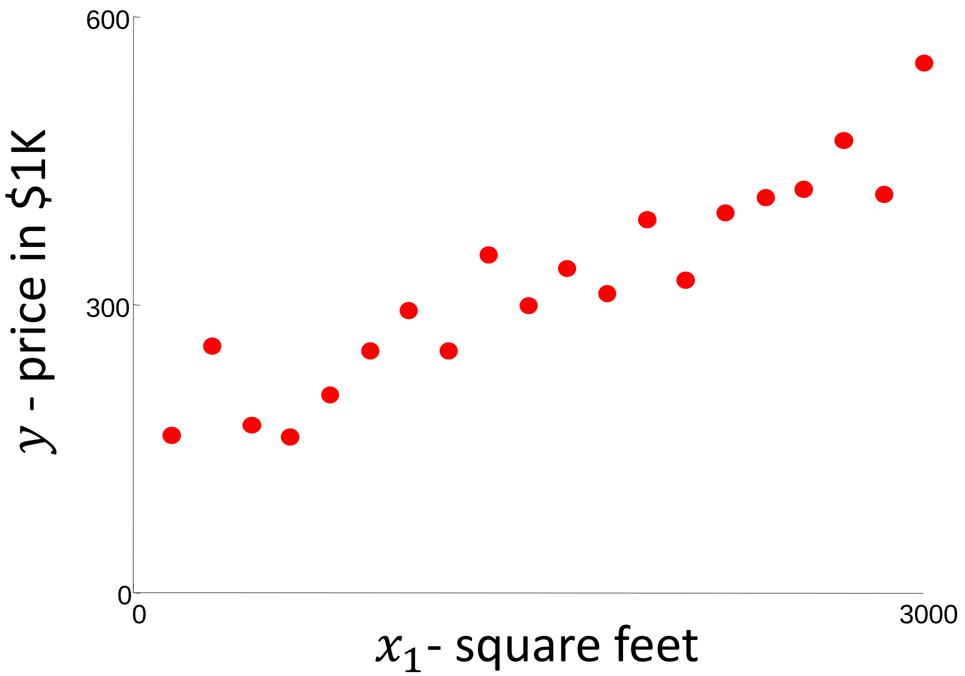
Supervised learning problems

- Supervised learning
  - We are given a training set, consisting of a list of m training examples $\{(x^{(i)}, y^{(i)}), i = 1 \dots m\}$
  - Our goal is to learn a function $h : \mathcal{X} \mapsto \mathcal{Y}$, so that $y^{(i)} \approx h(x^{(i)})$ for each training example
  - h *generalizes* well if it correctly predicts y's for novel examples (test set)
- When y is continuous, the learning problem is called a regression problem
  - E.g. predict the price of a house
- When y takes discrete values, the learning problem is a classification problem
  - E.g. predict the labels of digit images

Linear Regression

Linear regression

- Housing example
  - Assume one input feature, i.e. $x^{(i)} \in \mathbb{R}$. So, $\mathcal{X} = \mathcal{Y} = \mathbb{R}$

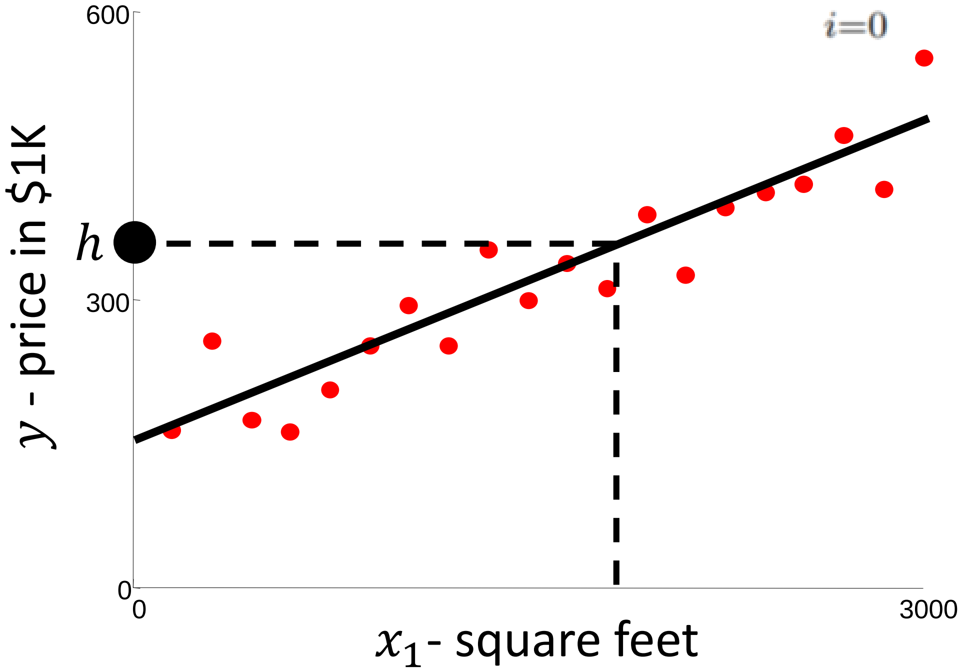


Living area (feet^2)	Price (\$1000s)
2100	410
1650	320
2200	387
…	…

Linear regression

- Housing example
  - Assume one input feature, i.e. $x^{(i)} \in \mathbb{R}$. So, $\mathcal{X} = \mathcal{Y} = \mathbb{R}$
  - We want to find a function $h : \mathcal{X} \mapsto \mathcal{Y}$. There are many choices!
  - A simple choice is to use a linear function parametrized by weights w:

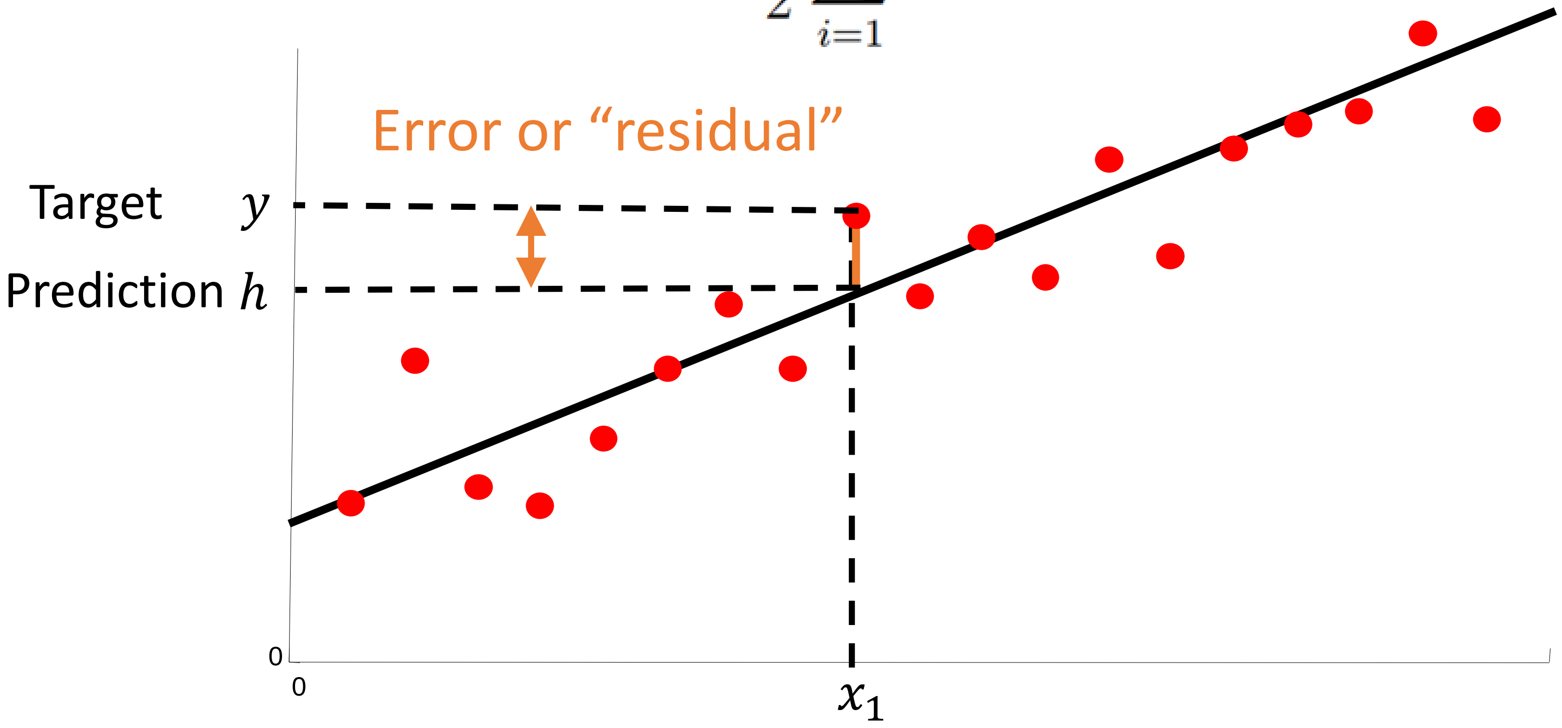
$$h_w(x) = w_0 + w_1 x_1 = \sum_{i=0}^{n} w_i x_i = w^T x$$



Least squares cost function

- Learning the parameters w?
  - Find a choice of w so that $h_w(x^{(i)}) \approx y^{(i)}$ for each i
  - To do so, we'll search for the parameters w that minimize the cost function:

$$J(w) = \frac{1}{2}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2$$

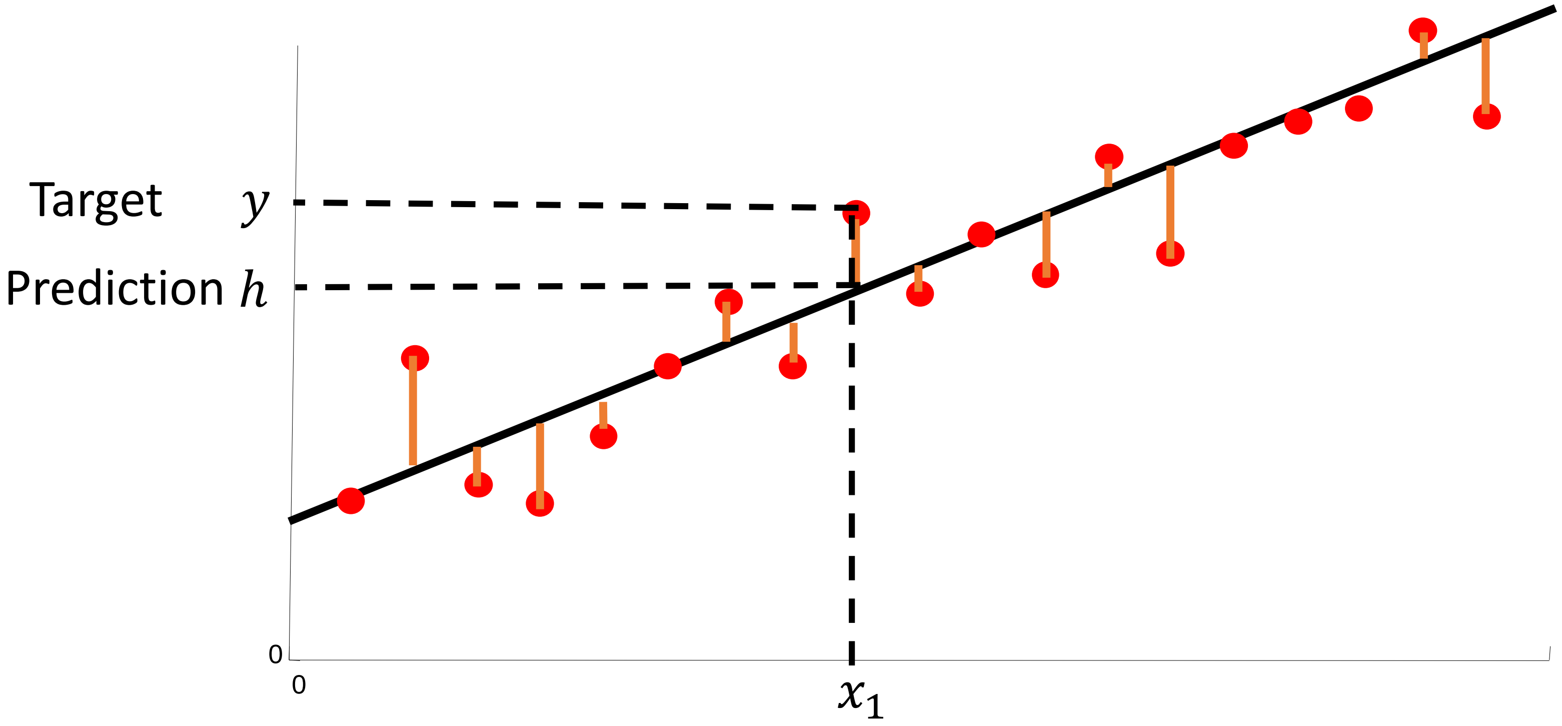


Least squares cost function

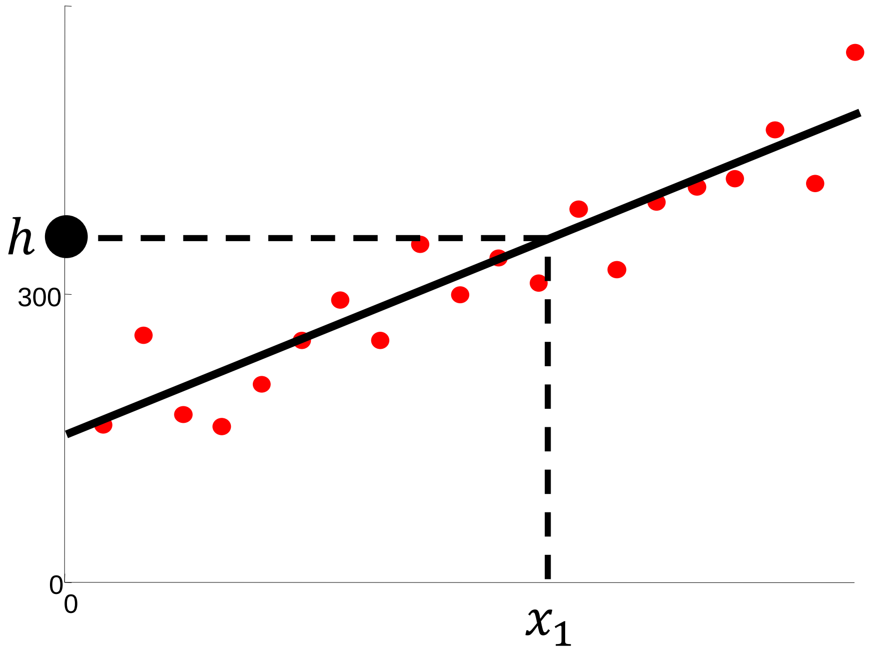
- Search for w that minimizes the following cost/loss/objective function:

$$J(w) = \frac{1}{2}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2$$

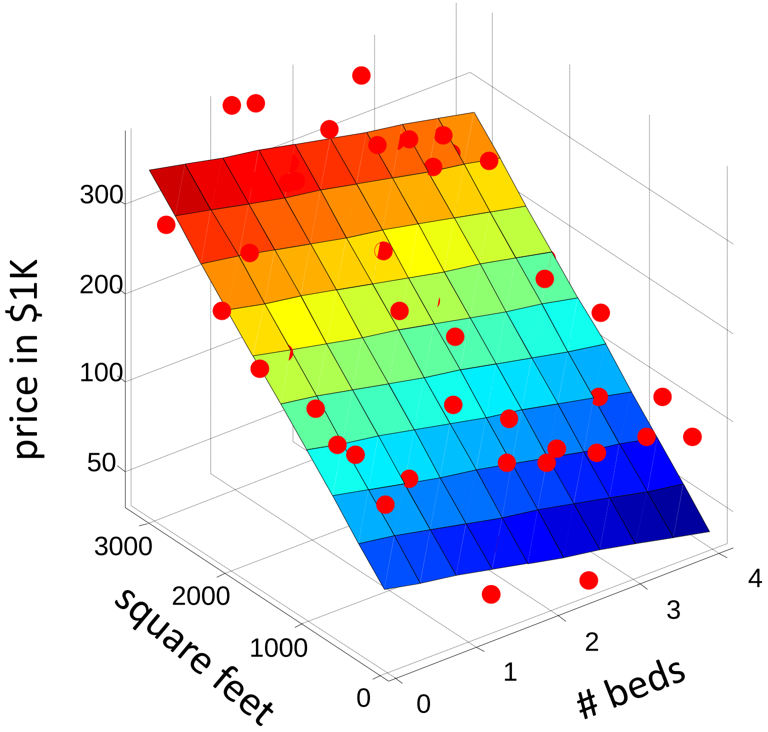
- This is known as least squares cost function



Linear regression



$h = w_0 + w_1 x_1$



$h = w_0 + w_1 x_1 + w_2 x_2$

Optimization

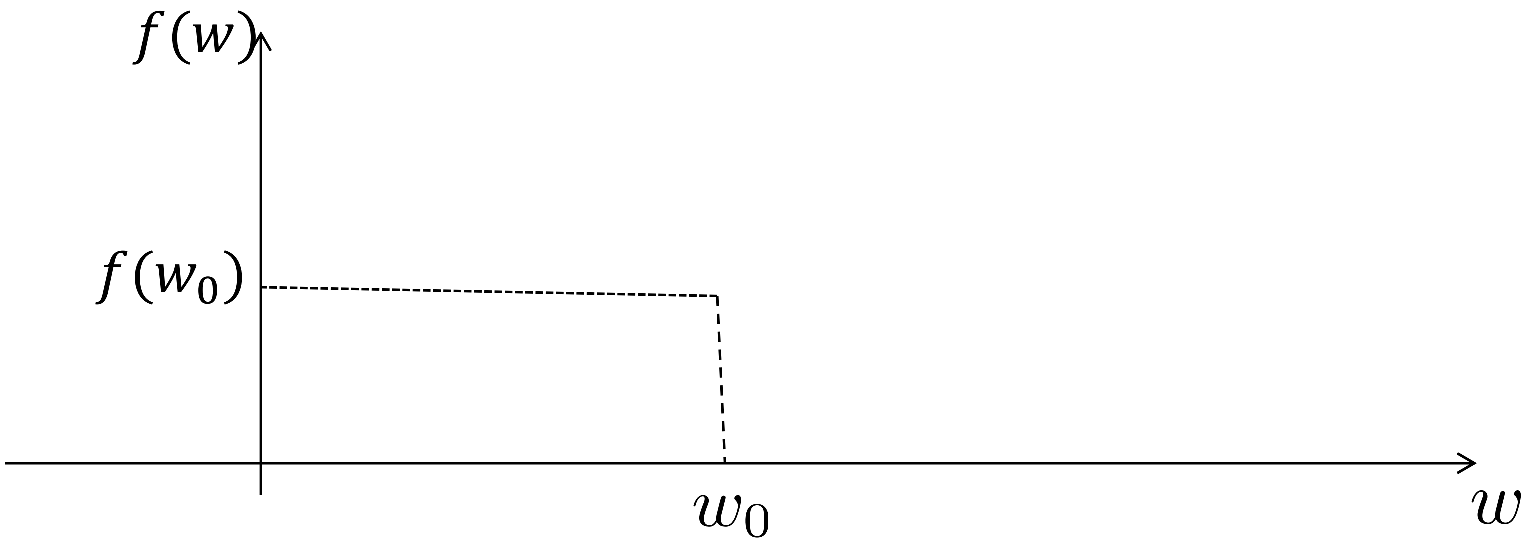
- How do we solve

$$\min_{w} J(w) = \min_{w} \frac{1}{2}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2$$

- or any $J(w)$ in that matter?

Gradient Descent

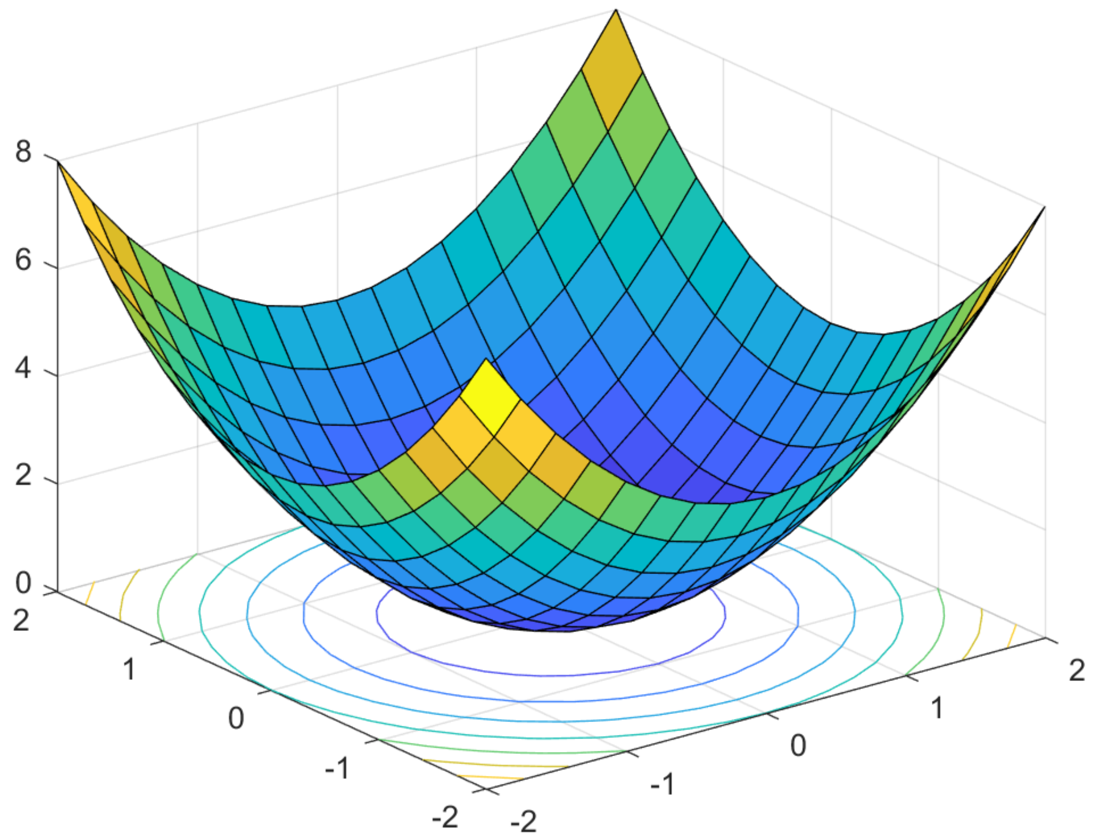
1-D optimization

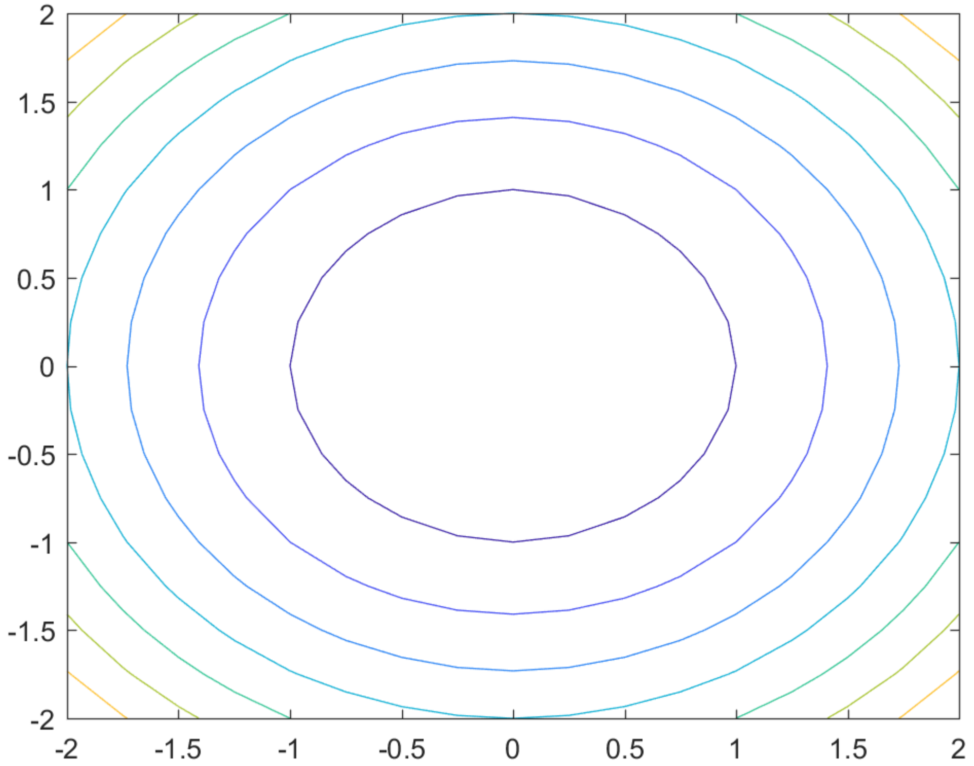


- Could evaluate $f(w_0 + \delta)$ and $f(w_0 - \delta)$
  - Then step in best direction
- Or, evaluate derivative:
  - Tells which direction to step into

$$\frac{df(w_0)}{dw} = \lim_{\delta \to 0} \frac{f(w_0 + \delta) - f(w_0 - \delta)}{2\delta}$$

2-D optimization



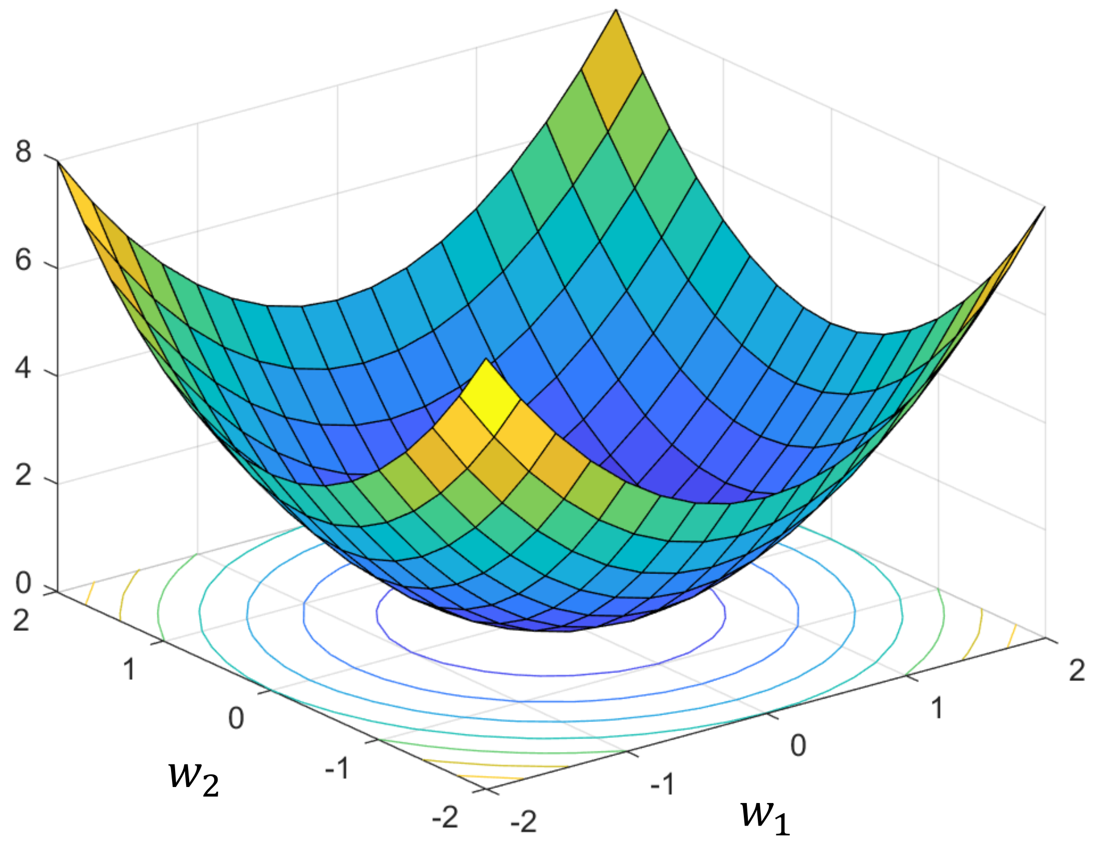


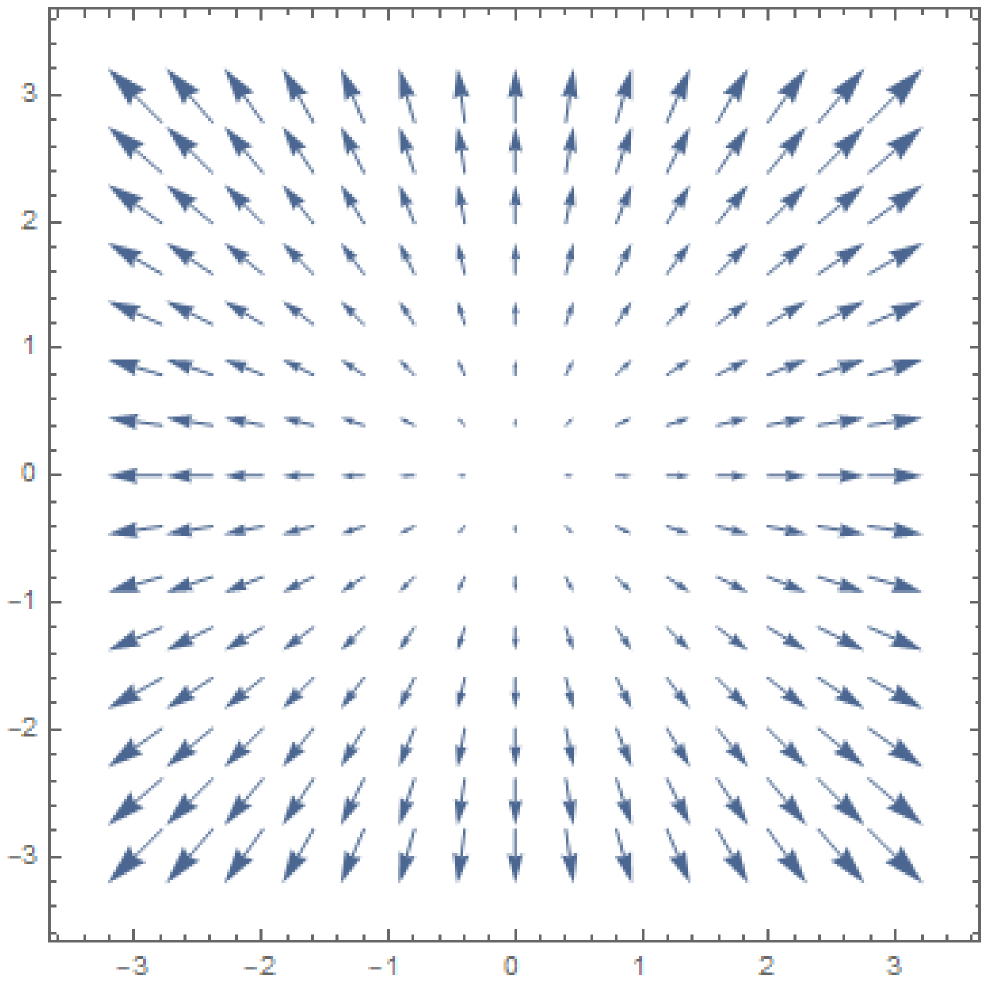
Gradient

- Let f be a multivariate function $f : \mathbb{R}^n \to \mathbb{R}$
- Its gradient is given by

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial w_1} \\ \frac{\partial f}{\partial w_2} \\ \cdots \\ \frac{\partial f}{\partial w_n} \end{bmatrix}$$

Gradient example





Gradient descent

- Perform update in downhill direction for each coordinate
- The steeper the slope (i.e. the higher the derivative) the bigger the step for that coordinate
- E.g., consider $f(w_0, w_1)$

  - Updates

$$w_1 \leftarrow w_1 - \alpha * \frac{\partial}{\partial w_1} f(w_1, w_2)$$

$$w_2 \leftarrow w_2 - \alpha * \frac{\partial}{\partial w_2} f(w_1, w_2)$$

  - Updates in vector notation

$$w \leftarrow w - \alpha * \nabla_w f(w)$$

$$\nabla_w f(w) = \begin{bmatrix} \frac{\partial}{\partial w_1} f(w) \\ \frac{\partial}{\partial w_2} f(w) \end{bmatrix}$$ (**gradient**)

Gradient descent

- Approach
  - Start somewhere
  - Repeat: Take a step proportional to the negative gradient direction

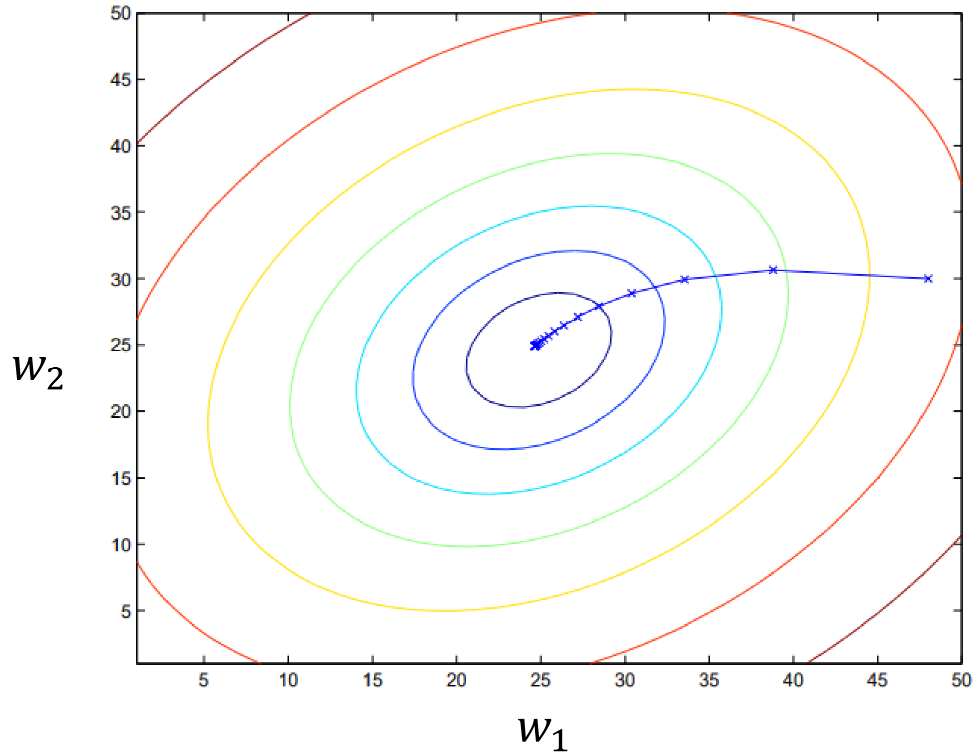
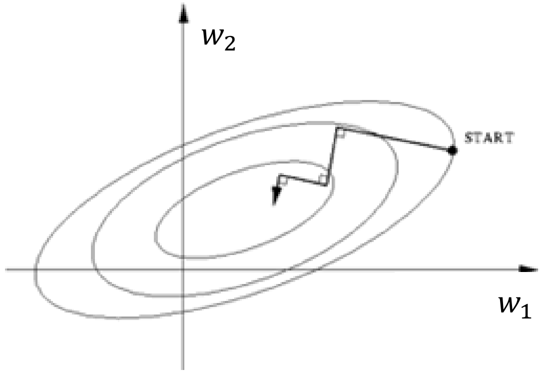


Figure source: Andrew Ng – CS229 Lecture Notes

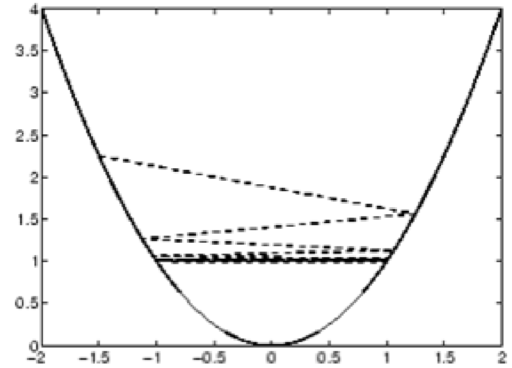
Optimization algorithm: Gradient Descent

- Let $f(w_1, \ldots, w_n)$ be a multivariate objective function
- To find a local minimum

> - `init` w
> - `for iter = 1,2,…`
>
> $w \leftarrow w - \alpha * \nabla f(w)$



- How to choose the learning rate α?
  - Try multiple choices, crude rule of thumb: update changes w about 0.1 – 1%
  - Linesearch: keep walking in the same direction as long as f is still decreasing



Batch gradient descent on least squares objective

$$\min_w J(w) = \quad \min_w \frac{1}{2}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2$$

- init w
- for iter = 1,2,…

$$w \leftarrow w - \alpha * \frac{1}{2}\sum_{i=1}^{m}\nabla(h_w(x^{(i)}) - y^{(i)})^2$$

Stochastic gradient descent on least squares objective

$$\min_w J(w) = \quad \min_w \frac{1}{2}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2$$

Idea: Once gradient on one training example has been computed, might as well incorporate before computing next one

- `init` w
- `for iter = 1,2,…`
  - `pick random k`

$$w \leftarrow w - \alpha * \frac{1}{2}\nabla(h_w(x^{(k)}) - y^{(k)})^2$$

Mini-batch gradient descent on least squares objective

$$\min_w J(w) = \min_w \frac{1}{2}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2$$

Idea: Gradient over small set of training examples (=mini-batch) can be computed in parallel, might as well do that instead of a single one

- init w
- for iter = 1,2,…
  - pick random subset of training examples B

$$w \leftarrow w - \alpha * \frac{1}{2}\sum_{b\in\mathbf{B}} \nabla(h_w(x^{(b)}) - y^{(b)})^2$$

How about the gradient of the least squares?

- Need to compute $\nabla_w (h_w(x) - y)^2$
- Let's focus on a specific coordinate *j*

$$\begin{aligned}
\frac{\partial}{\partial w_j}(h_w(x) - y)^2 &= 2 \cdot (h_w(x) - y) \cdot \frac{\partial}{\partial w_j}(h_w(x) - y) \\
&= 2 \cdot (h_w(x) - y) \cdot \frac{\partial}{\partial w_j}\left(\sum_i^n w_i x_i - y\right) \\
&= 2 \cdot (h_w(x) - y) x_j
\end{aligned}$$

Linear regression update rules

- Batch gradient descent (for every *j*)

$$w_j \leftarrow w_j - \alpha \sum_{i=1}^{m} (h_w(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

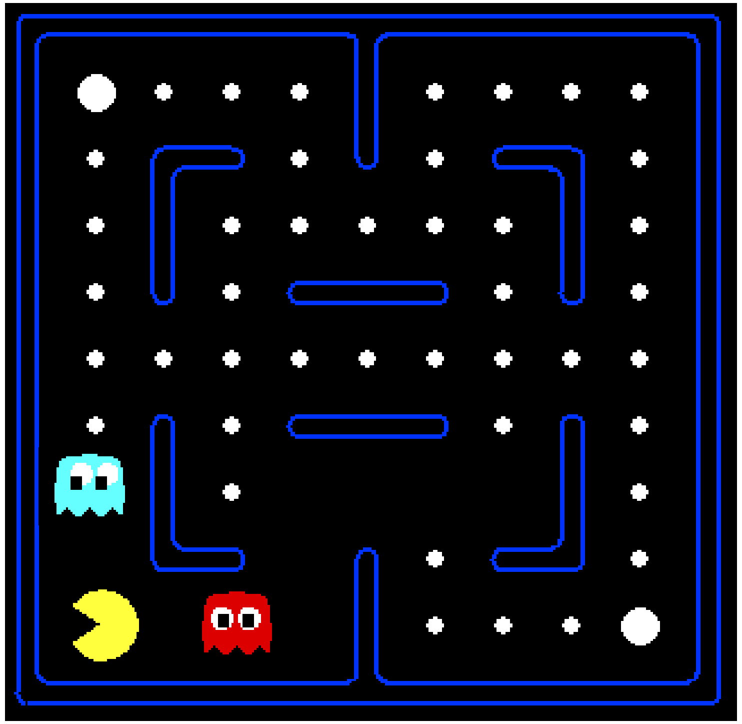
- Stochastic gradient descent (for every *j*)

$$w_j \leftarrow w_j - \alpha \, (h_w(x^{(k)}) - y^{(k)}) x_j^{(k)}$$

Revisiting Approximate Q-Learning

Recall: Approximate Q-learning

- Describe a q-state using a vector of features
  - Example features:
    - Distance to closest ghost
    - Distance to closest dot
    - Number of ghosts
    - Is Pacman in dead-end?
    - ……



Approximate Q-learning & supervised learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Q-learning with linear functions
  - Receive a sample transition (s, a, r, s')
  - Consider the sample estimate

    $$\textit{target}(s') = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Update the weights

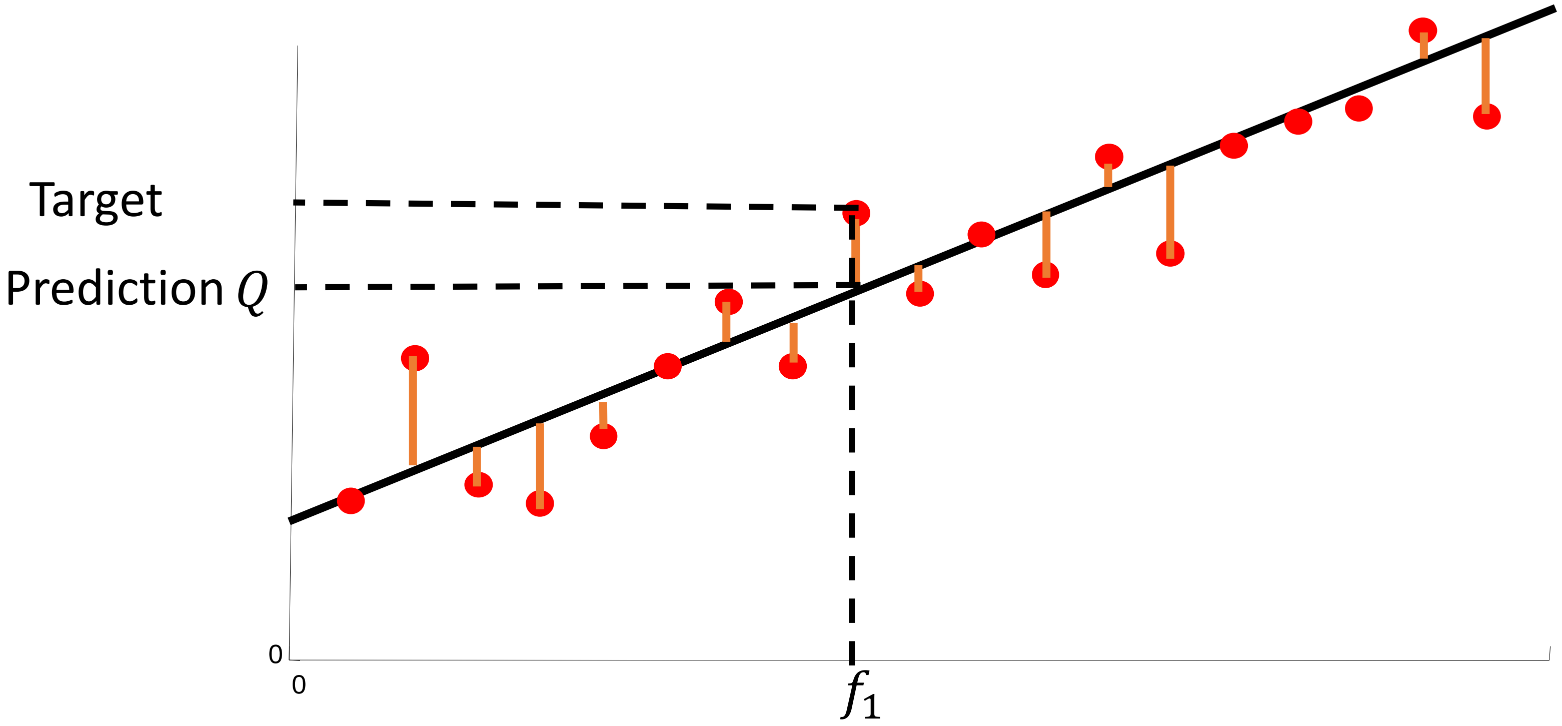
    $$w_j \leftarrow w_j + \alpha[\, target(s') - Q(s,a)\,]\, f_j(s,a)$$

Recall: Least squares cost function

- Search for w that minimizes the following cost/loss/objective function:

$$J(w) = \frac{1}{2}\sum_{i=1}^{m}(h_w(x^{(i)}) - y^{(i)})^2$$

- This is known as least squares cost function



Approximate Q-learning & supervised learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Q-learning with linear functions
  - Receive a sample transition (s, a, r, s')
  - Consider the sample estimate

    $$\textit{target}(s') = R(s,a,s') + \gamma \max_{a'} Q(s',a')$$

  - Update the weights

    $$w_j \leftarrow w_j + \alpha[\, target(s') - Q(s,a)\,]\, f_j(s,a)$$

- Interpretation
  - Stochastic gradient descent on a least square cost function

Minimizing error

Consider only one sample, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial \text{ error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

Minimizing error

Consider only one sample, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\ \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha \left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

Approximate Q update explained:

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$$

“target” “prediction”

Other function approximations

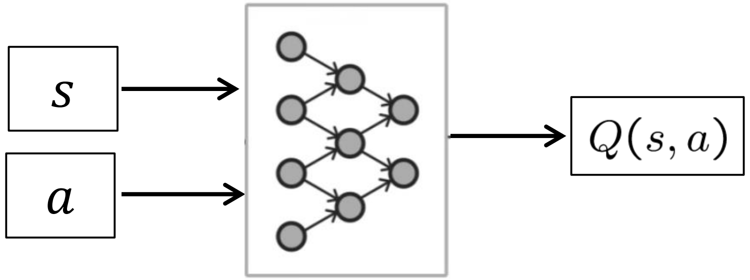
- Instead of a table, we have parametrized Q (or V) function: $Q_w(s, a)$
  - It can be a linear function in features f_i

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

  - Or some other function, e.g., polynomial

$$Q(s,a) = w_{11} f_1(s,a) + w_{12} f_1(s,a)^2 + w_{13} f_1(s,a)^3 + \ldots$$

  - Or a neural network (learn the features f_i too!)



- Update rule

$$w_m \leftarrow w_m + \alpha \left[r + \gamma \max_a Q(s', a') - Q(s,a) \right] \frac{dQ}{dw_m}(s,a)$$