Linear Regression and Backpropagation

AI+X
INSTITUTE FOR ARTIFICIAL INTELLIGENCE

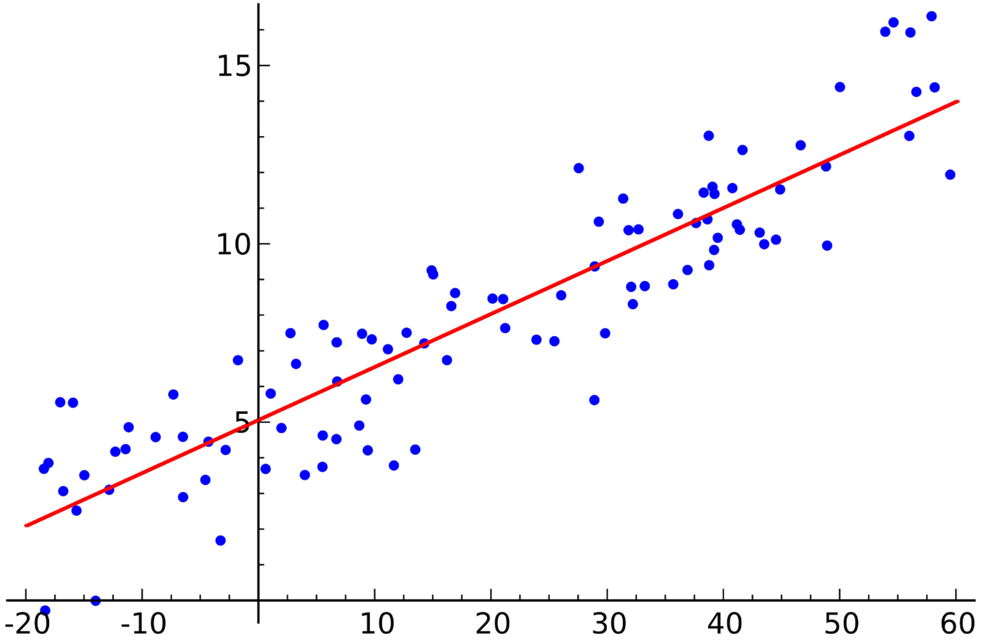
Linear Regression

- Fit a straight line or surface into an existing dataset X = {(X_1,Y_1), …, (X_N,Y_N)} in a way that minimizes the discrepancies between predicted and expected values
- In the simplest case, we want to regress one value from one input value:

$$\hat{y}_i^* = a^* x_i + b^*$$

$$(a^*, b^*) = \text{argmin}_{a,b} \sum_i (\hat{y}_i - y_i)^2 =$$

$$\text{argmin}_{a,b} \sum [(a x_i + b) - y_i]^2$$

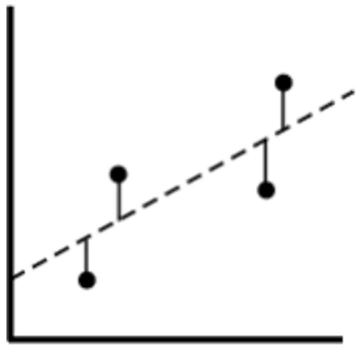


Linear Regression

- The loss function $\mathcal{L}(X)$ measures the vertical deviations between the predicted values $\hat{y}_i$ and expected values y_i

$$\hat{y}_i = ax_i + b$$

$$\mathcal{L}(X) = \sum_i(\hat{y}_i - y_i)^2 = \sum[(ax_i + b) - y_i]^2$$



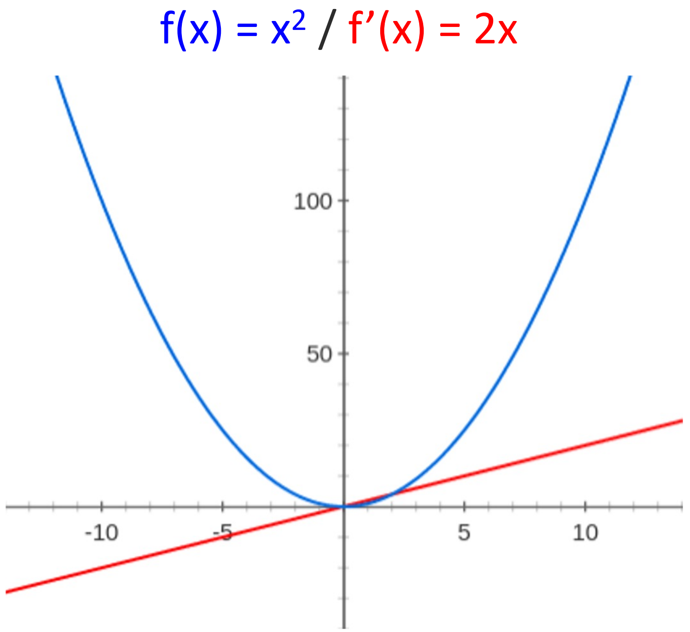
Linear Regression

The loss function:

$$\mathcal{L} = \sum_i[(ax_i + b) - y_i]^2$$

is minimized when:

$$\frac{\partial \mathcal{L}}{\partial a} = 0, \quad \text{and} \quad \frac{\partial \mathcal{L}}{\partial b} = 0$$



Linear Regression – Least Square Fitting

The loss function:

$$\mathcal{L} = \sum_i[(ax_i + b) - y_i]^2$$

is minimized when:

$$\frac{\partial\mathcal{L}}{\partial a} = 0, \qquad \text{and} \qquad \frac{\partial\mathcal{L}}{\partial b} = 0.$$

These lead to the equations:

$$\frac{\partial\mathcal{L}}{\partial b} = 2\sum_i[(ax_i + b) - y_i] = 0$$

$$= a\sum_i x_i + Nb = \sum_i y_i$$

$$\frac{\partial\mathcal{L}}{\partial a} = 2\sum_i[(ax_i + b) - y_i]x_i = 0$$

$$= a\sum_i x_i^2 + b\sum_i x_i = \sum_i x_i y_i$$

In matrix form:

$$\begin{vmatrix} \sum_i x_i & N \\ \sum_i x_i^2 & \sum_i x_i \end{vmatrix} \begin{vmatrix} a \\ b \end{vmatrix} = \begin{vmatrix} \sum_i y_i \\ \sum_i x_i y_i \end{vmatrix}$$

$$\begin{vmatrix} a \\ b \end{vmatrix} = \begin{vmatrix} \sum_i x_i & N \\ \sum_i x_i^2 & \sum_i x_i \end{vmatrix}^{-1} \begin{vmatrix} \sum_i y_i \\ \sum_i x_i y_i \end{vmatrix}$$

In closed-form:

$$a = \sum_i(x_i - \bar{x})(y_i - \bar{y}) / \sum_i(x_i - \bar{x})^2$$

$$b = \bar{y} - a\bar{x}$$

Linear Regression – Least Square Fitting

```
def linear_least_squares_regression(points):
    X = points[:,0]
    Y = points[:,1]

    X_mean = np.mean(X)
    Y_mean = np.mean(Y)

    a = np.sum(np.multiply(X-X_mean, Y-Y_mean)) / np.sum(np.square(X-X_mean))
    b = Y_mean - a * X_mean

    return a, b
```

Linear Regression – Least Square Fitting

- Closed-form solution
- Impractical/impossible to adapt to more complicated functions

? Can we exploit the same idea in a simpler way?

Linear Regression

Initially guess the values of a and b. The average of the squares of the vertical deviations will be:

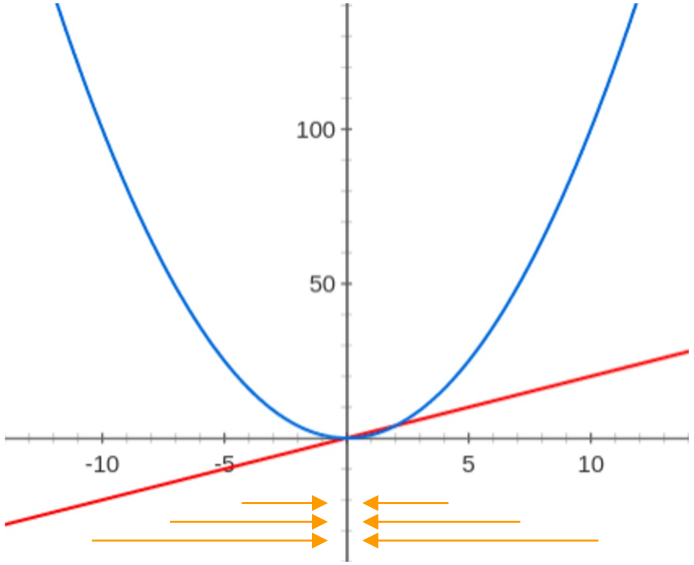
$$\mathcal{L} = 1/N \sum_i[(ax_i + b) - y_i]^2$$

Thus, we have the following partial derivatives:

$$\frac{\partial \mathcal{L}}{\partial b} = \nabla_b = 2/N \sum_i[(ax_i + b) - y_i]$$

$$\frac{\partial \mathcal{L}}{\partial a} = \nabla_a = 2/N \sum_i[(ax_i + b) - y_i]x_i$$

$f(x) = x^2$ / $f'(x) = 2x$



$-\nabla$

Linear Regression – Gradient Descent

Initially guess the values of a and b. The average of the squares of the vertical deviations will be:

$$\mathcal{L} = 1/N \sum_i [(ax_i + b) - y_i]^2$$

The partial derivatives:

$$\frac{\partial \mathcal{L}}{\partial b} = \nabla_b = 2/N \sum_i [(ax_i + b) - y_i]$$

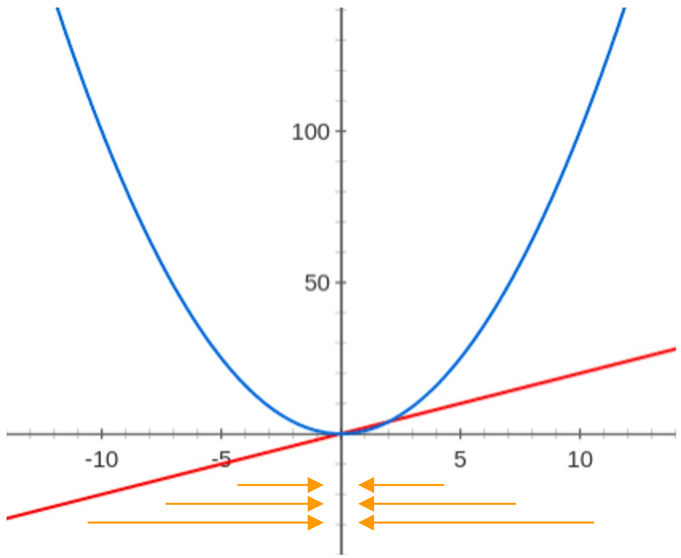
$$\frac{\partial \mathcal{L}}{\partial a} = \nabla_a = 2/N \sum_i [(ax_i + b) - y_i]x_i$$

will indicate how to update a and b to obtain a better fitting result:

$$b_{t+1} = b_t - \lambda \nabla_b$$
$$a_{t+1} = a_t - \lambda \nabla_a$$

where λ is the learning rate.

$f(x) = x^2$ / $f'(x) = 2x$



$-\nabla$

Linear Regression – Gradient Descent

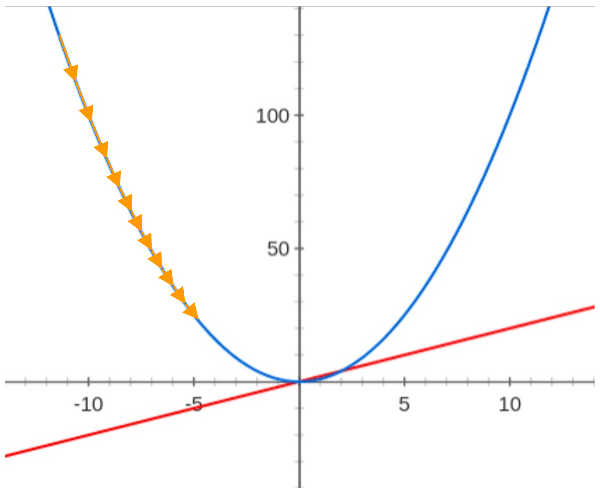
```
def gradient_descent_step(points, a, b, learning_rate):
  X = points[:,0]
  Y = points[:,1]

  tmp = ((a*X + b) - Y) * (2.0/len(points))
  a_grad = np.sum(np.multiply(tmp, X))
  b_grad = np.sum(tmp)

  a_ = a - learning_rate * a_grad
  b_ = b - learning_rate * b_grad

  return a_, b_
```

Common Practice #1 – Hyperparameter Tuning

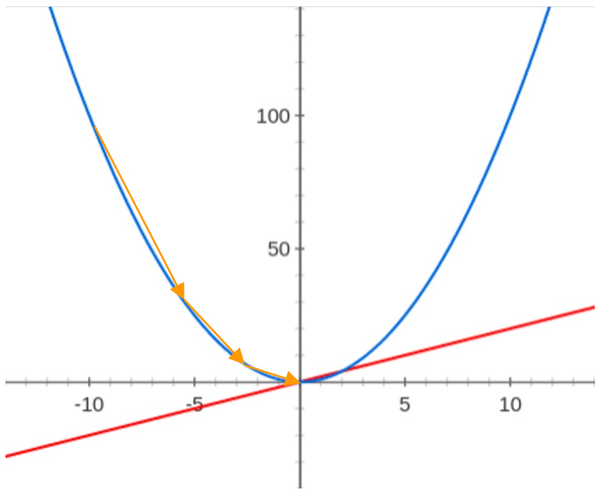
- Repeat the training several times with different hyperparameter values
- Grid search
  - Exhaustive search on a predefined set of hyperparameter values
  - Ex: learning_rate → {1.0, 0.1, 0.01, 0.001}
- Ideally, the optimal value is not in the edge of the set

Gradient Descent – Picking a Learning Rate λ

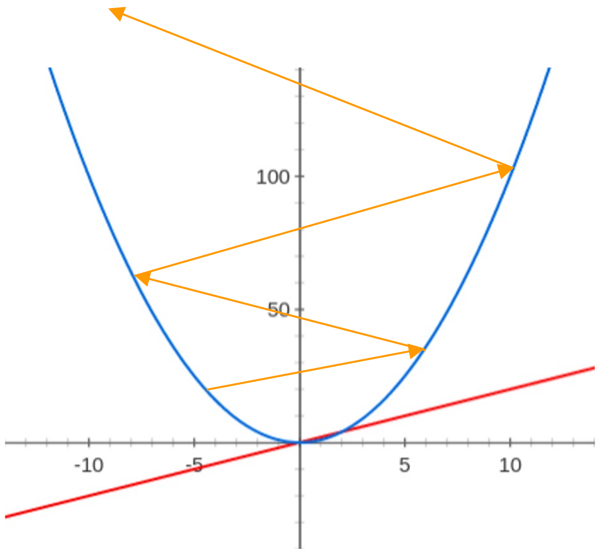
- It is a common practice to test different learning rates within a predefined range and pick the one that converges faster



Too small



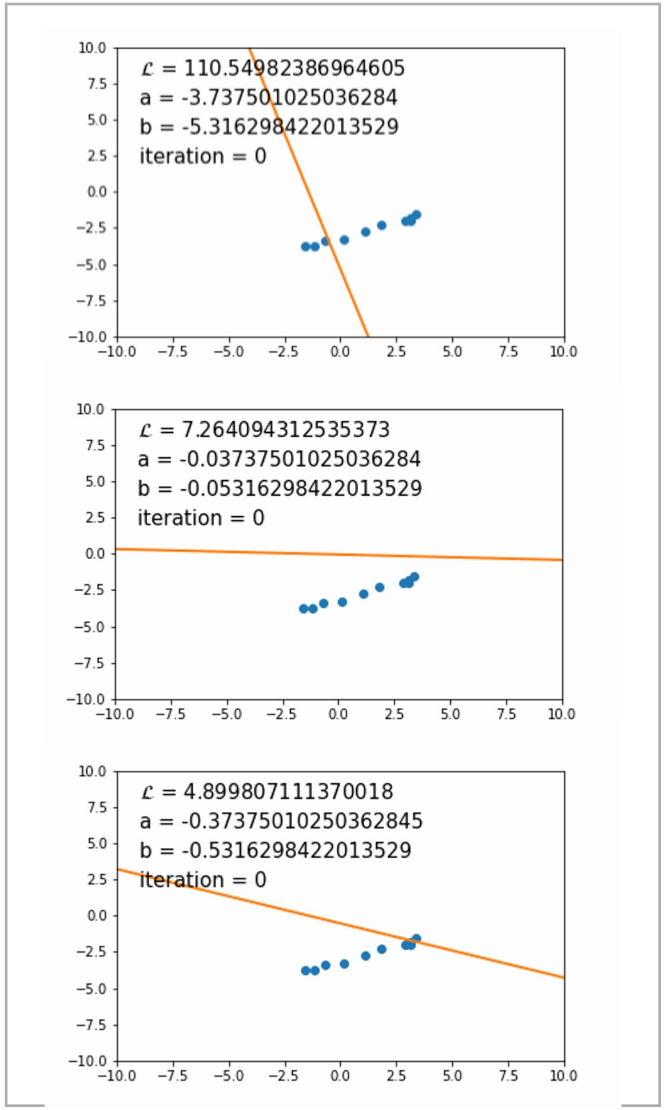
Adequate



Too large

Knowledge Check 1

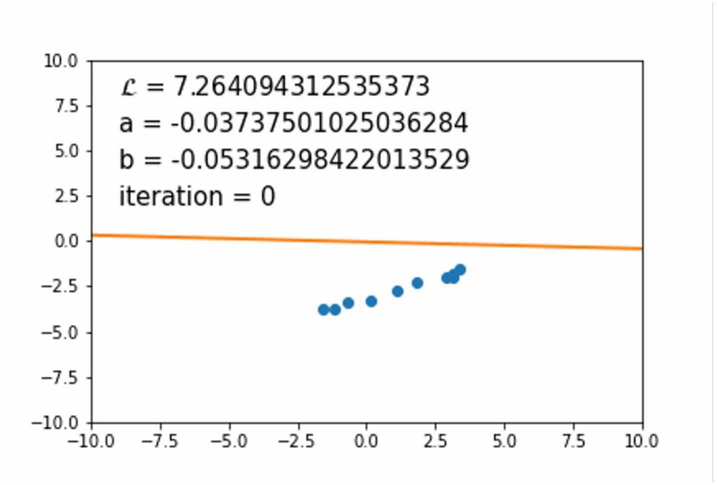
Which alternative best describes the learning rate for the regression scenarios shown in the right, from top to bottom?



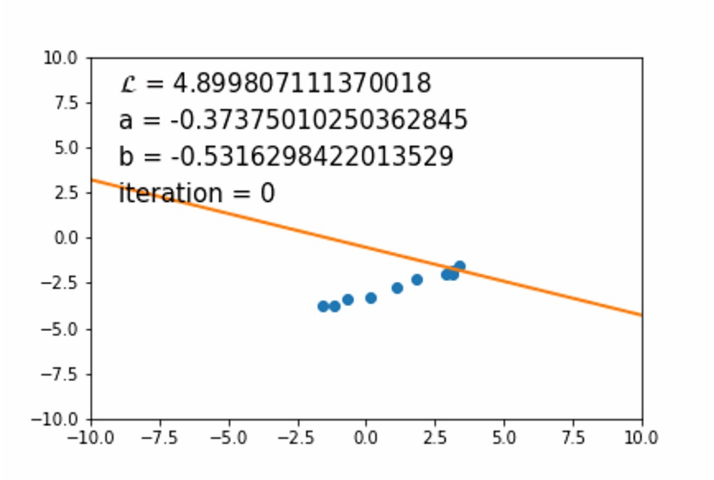
A	Too low, adequate, too large
B	Too low, too large, adequate
C	Adequate, too low, too large
D	Adequate, too large, too low
E	Too large, too low, adequate
F	Too large, adequate, too low

Gradient Descent – Picking a Learning Rate λ

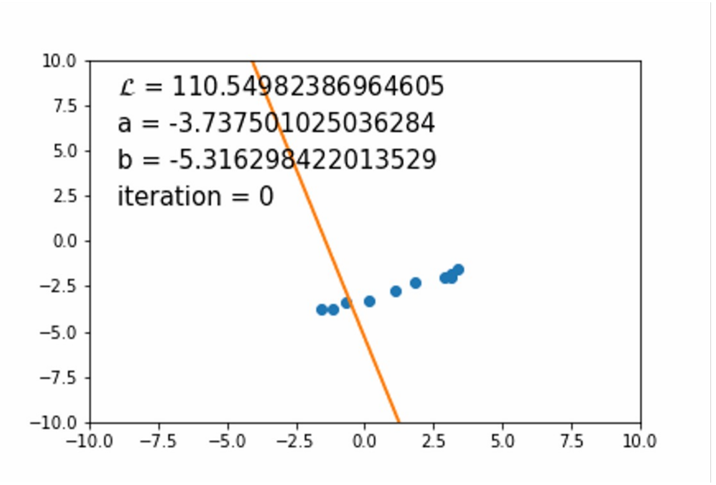
- Picking the best λ value depends on different factors (loss, function, data)



Too small (10^{-2})



Adequate (10^{-1})



Too large (10^{0})

Linear Regression – Multivariate Data

- Input may have multiple values
- Output may have multiple values

$$\hat{\mathbf{y}} = W\mathbf{x} + \mathbf{b}$$

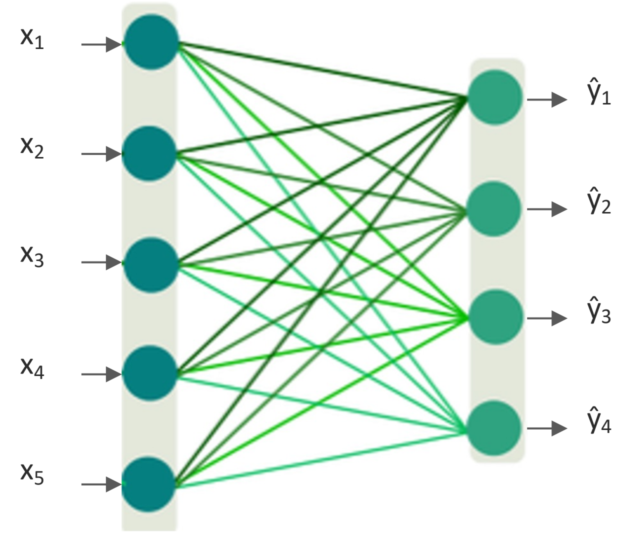
$$\hat{y}_j = w_{1j}x_1 + w_{2j}x_2 + \ldots + w_{|\mathbf{x}|j}x_{|\mathbf{x}|} + b_j$$

$$\mathcal{L} = (½)\sum_k(\hat{y}_k - y_k)^2$$

Number of parameters:

$$|W| = |\mathbf{x}| * |\mathbf{y}|$$

$$|b| = |\mathbf{y}|$$



Linear Regression – Multivariate Data

$$w_{ij}^{t+1} = w_{ij}^{t} - \lambda \nabla_{w[ij]} = w_{ij}^{t} - \lambda \frac{\partial \mathcal{L}}{\partial w_{ij}}$$

$$b_{j}^{t+1} = b_{j}^{t} - \lambda \nabla_{b[j]} = b_{j}^{t} - \lambda \frac{\partial \mathcal{L}}{\partial b_{j}}$$

Linear Regression – Multivariate Data

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = \frac{\partial}{\partial w_{ij}}(½)\sum_k(\hat{y}_k - y_k)^2 = \frac{\partial}{\partial w_{ij}}(½)(\hat{y}_j - y_j)^2 =$$

$$(\hat{y}_j - y_j)\frac{\partial}{\partial w_{ij}}(\hat{y}_j - y_j) = (\hat{y}_j - y_j)\frac{\partial}{\partial w_{ij}}\mathbf{x}\mathbf{w}_{*\mathbf{j}} + b_j - y_j =$$

$$(\hat{y}_j - y_j)\frac{\partial}{\partial w_{ij}}\mathbf{x}\mathbf{w}_{*\mathbf{j}} = (\hat{y}_j - y_j)x_i$$

Linear Regression – Multivariate Data

$$\frac{\partial \mathcal{L}}{\partial b_j} = \frac{\partial}{\partial b_j}(½)\sum_k(\hat{y}_k - y_k)^2 = \frac{\partial}{\partial b_j}(½)(\hat{y}_j - y_j)^2 =$$

$$(\hat{y}_j - y_j)\frac{\partial}{\partial b_j}(\hat{y}_j - y_j) = (\hat{y}_j - y_j)\frac{\partial}{\partial b_j}\mathbf{x}\mathbf{w}_{*j} + b_j - y_j =$$

$$(\hat{y}_j - y_j)\frac{\partial}{\partial b_j}b_j = (\hat{y}_j - y_j)$$

Linear Regression – Multivariate Data

$$\frac{\partial \mathcal{L}}{\partial w_{ij}} = (\hat{y}_j - y_j)x_i$$

$$\frac{\partial \mathcal{L}}{\partial b_j} = (\hat{y}_j - y_j)$$

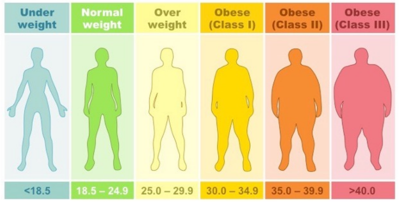
Linear Regression – Multivariate Data

- x = (height, weight) = (x_0, x_1)
- y = BMI
- $\hat{y} = w_0 x_{i0} + w_1 x_{i1} + b$
- One gradient per weight:

$$\frac{\partial \mathcal{L}}{\partial b} = \nabla_b = (w_0 x_0 + w_1 x_1 + b) - y$$

$$\frac{\partial \mathcal{L}}{\partial w_0} = \nabla_{w[0] = [(} w_0 x_0 + w_1 x_1 + b) - y] x_0$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \nabla_{w[1] = [(} w_0 x_0 + w_1 x_1 + b) - y] x_1$$



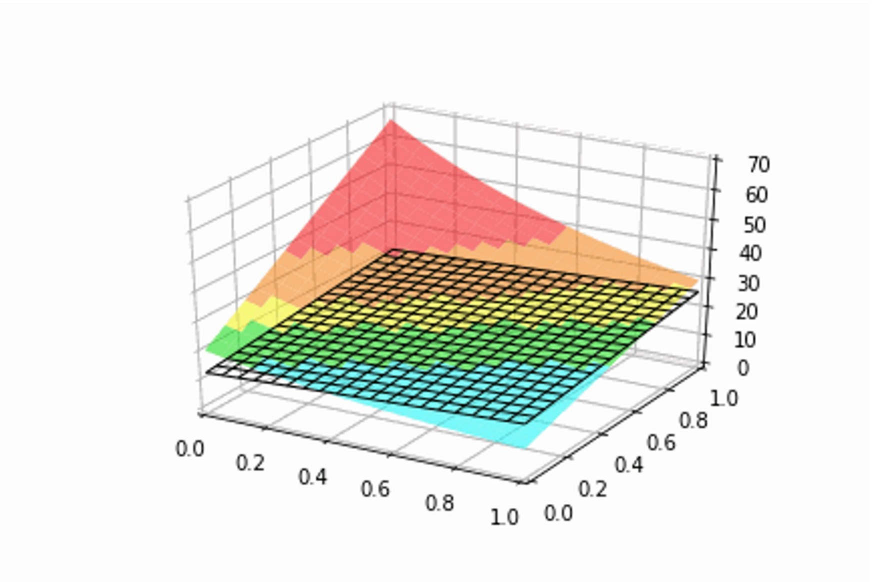
WEIGHT

	lbs	90	100	110	120	130	140	150	160	170	180	190	200	210	220	230	240	250	260	270	280	290
	kgs	41	45	50	54	59	64	68	73	77	82	86	91	95	100	104	109	113	118	122	127	132
ft/in	cm																					
4’8”	142.2	20	22	25	27	29	31	34	36	38	40	43	45	47	49	52	54	56	58	61	63	65
4’9”	144.7	19	22	24	16	28	30	32	35	37	39	41	43	45	48	50	52	54	56	58	61	63
4’10”	147.3	19	21	19	25	27	29	31	33	36	38	40	42	44	46	48	50	52	54	56	59	61
4’11”	149.8	18	20	22	24	26	28	30	32	34	26	38	40	42	44	46	48	51	53	55	57	59
5’0”	152.4	18	20	21	23	25	27	29	31	33	35	37	39	41	43	45	47	49	51	53	55	57
5’1”	154.9	17	19	21	22	25	26	28	30	32	34	36	39	40	42	43	45	47	49	51	53	55
5’2”	157.4	16	18	20	22	24	26	27	29	31	33	35	37	38	20	42	44	46	48	49	51	53
5’3”	160.0	16	18	19	21	23	25	27	28	30	32	34	35	37	39	41	43	44	46	48	50	51
5’4”	162.5	15	17	19	21	22	24	26	27	29	31	33	34	36	27	39	41	43	45	46	48	50
5’5”	165.1	15	17	18	20	22	23	25	27	28	30	32	33	35	27	38	40	42	43	45	47	49
5’6”	167.6	15	16	18	19	21	23	24	26	27	29	31	32	34	26	37	39	40	42	44	45	47
5’7”	170.1	14	16	17	19	20	22	24	25	27	29	30	31	33	25	36	38	39	41	42	44	45
5’8”	172.7	14	15	17	18	20	21	23	24	26	27	29	30	32	24	35	37	38	40	41	43	44
5’9”	175.2	13	15	16	18	19	21	22	24	25	27	28	30	31	24	34	35	37	39	40	41	43
5’10”	177.8	13	14	16	17	19	20	22	23	24	26	27	29	30	23	33	34	36	37	39	40	42
5’11”	180.3	13	14	15	14	18	20	21	22	24	25	27	28	29	22	32	33	35	36	38	39	40
6’0”	182.8	12	14	15	16	18	16	20	22	23	24	26	27	28	22	31	33	34	35	37	38	39
6’1”	185.4	12	13	15	16	17	18	20	21	22	24	25	26	28	21	30	32	33	34	36	37	39
6’2”	187.9	12	13	14	15	17	18	19	21	22	23	24	26	27	21	30	31	32	33	35	36	37
6’3”	190.5	11	13	14	15	16	17	19	20	21	23	24	25	26	20	29	30	31	32	34	35	36
6’4”	193.0	11	12	13	15	16	17	18	19	21	22	23	24	26	19	28	29	30	32	33	34	35
6’5”	195.5	11	12	13	14	15	17	18	19	20	21	23	24	25	19	27	28	30	31	32	33	34
6’6”	198.1	10	12	13	14	15	16	17	18	20	21	22	23	24	18	27	28	29	30	31	32	34
6’7”	200.6	10	11	12	14	15	16	17	18	19	20	21	23	24	18	26	27	28	29	30	32	33
6’8”	203.2	10	11	12	13	14	15	16	18	19	20	21	22	23	18	25	26	27	28	30	31	32
6’9”	205.7	10	11	12	13	14	15	16	17	18	19	20	21	23	17	25	25	27	28	29	30	31
6’10”	208.2	9	10	12	13	14	14	16	17	18	19	20	21	22	17	24	25	26	27	28	29	30
6’11”	210.8	9	10	11	12	13	14	15	16	17	18	19	20	21	16	23	25	25	27	28	29	30

HEIGHT

Linear Regression – Gradient Descent for Multivariate Data

```
def gradient_descent_step(X, Y, W, b, learning_rate):
  tmp = (np.matmul(X, np.transpose(W)) + b - Y)
  W_grad = np.matmul(np.transpose(tmp), X) / X.shape[0]
  b_grad = np.mean(tmp, axis=0)

  W_ = W - learning_rate * W_grad
  b_ = b - learning_rate * b_grad

  return W_, b_
```



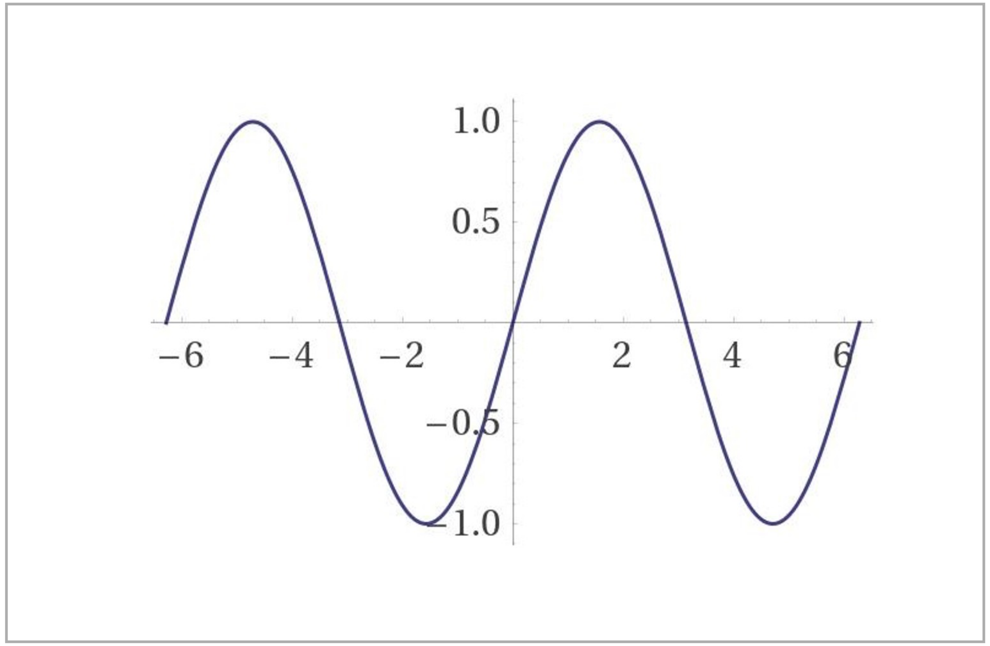
Common Practice #2 – Data Normalization

- Large values affect gradient-based optimization performance
- Differences in scale may increase the difficulty of the problem
- Gradients from larger parameters dominate the updates
- Poor generalization
- Common approaches:
  - Normalize inputs to the range from 0 to 1
  - Normalize inputs to the range from -1 to 1
  - Normalize inputs to an unit vector
  - Normalize inputs to a specific distribution (e.g. $\mathcal{N}(0,1)$)

Knowledge Check #2

Let's say you uniformly sample points from the function sin(x) in the range [-2π,2π] and use linear regression to fit a function f into those points. What will be the value of f(x)?

A	sin(x)
B	cos(x)
C	1
D	0
E	-1



You have reached the end of the lecture.