DATASCI207-005/007 Applied Machine Learning

Vilena Livinsky, PhD(c)

School of Information, UC Berkeley

Week 2: 01/15/2025 & 01/16/2025

Today's Agenda

- Logistics note: Final Project
- Linear Regression & Gradient Descent
- Walkthroughs:
  - Gradient Descent & Linear Regression
  - TensorFlow Intro

Final Project: Step 1

Form a group

- 3-4 people, max 4
- NO silos or groups of 2
- Groups can only be composed of you and your colleagues in your section

Inform me & the class of your formed group in Slack

- Include names of group members
- Due date: 01/24/2025 EOD

General Plan:

- Step 1: form a group
- Step 2: submit your group's question to investigate + dataset
- Step 3: a baseline presentation
- Step 4: final presentation

Walkthrough

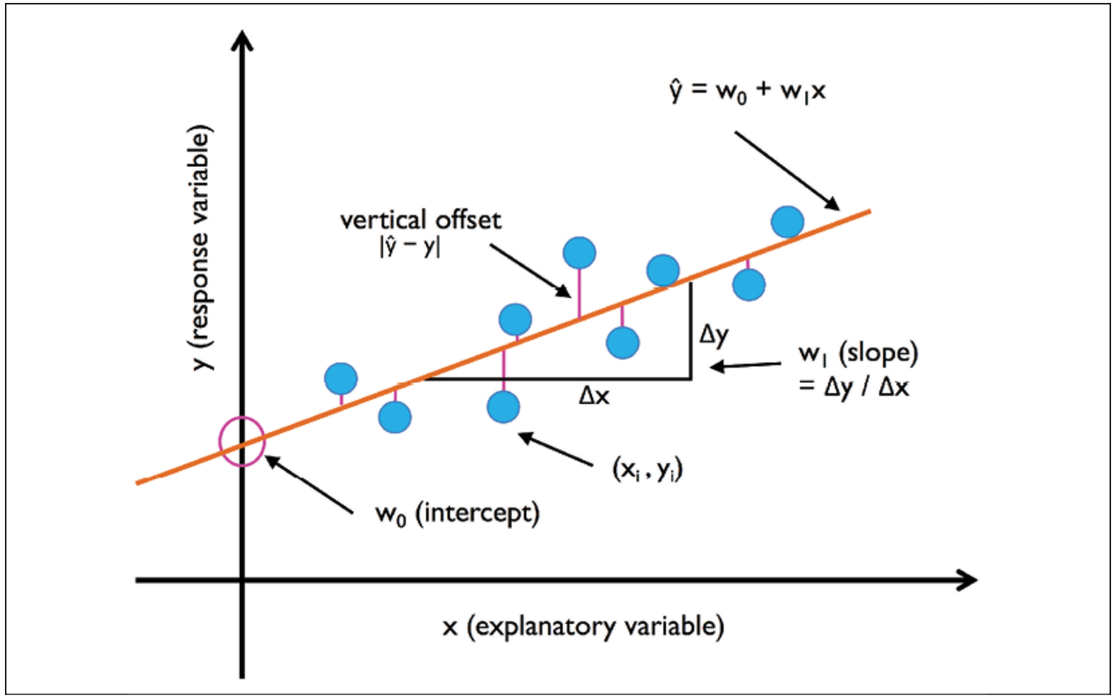
01b_Framing.ipynb

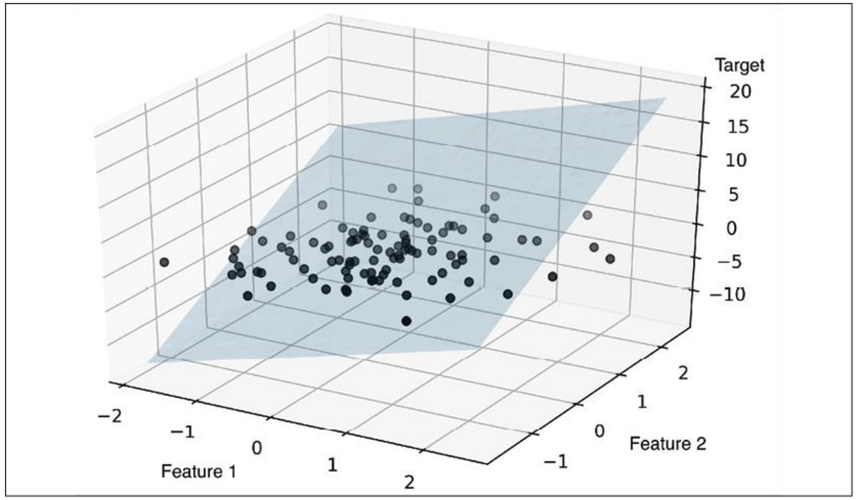
Linear Regression & Gradient Descent (Review)

- Supervised learning
- predict target variable on a continuous scale
- simple (univariate) linear regression vs. multivariate case

$$y = w_0x_0 + w_1x_1 + \dots + w_mx_m = \sum_{i=0}^{n} w_ix_i = w^Tx$$

Here, w_0 is the y axis intercept with $x_0 = 1$.





Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

Linear Regression & Gradient Descent (Review)

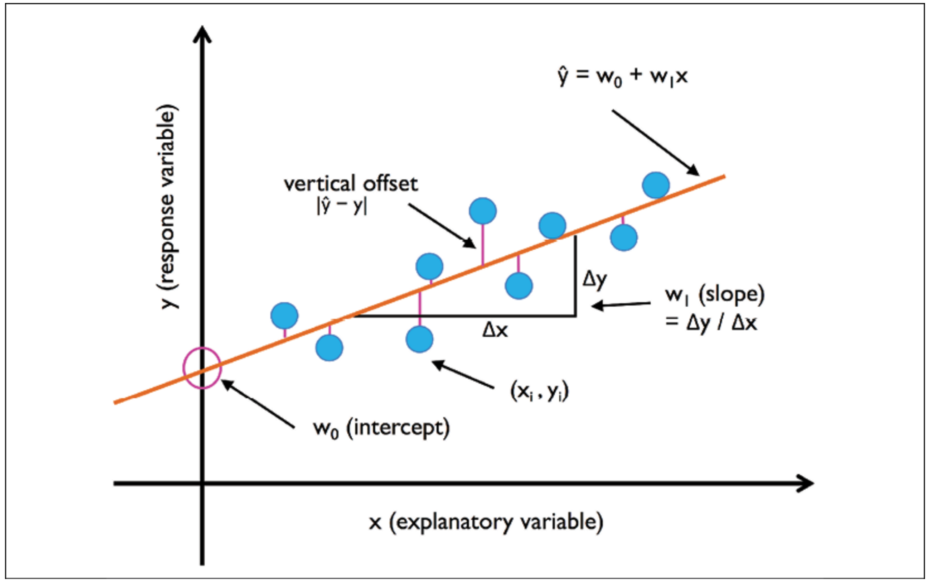
- Supervised learning
- predict target variable on a continuous scale
- simple (univariate) linear regression vs. multivariate case
- Gradient Descent:
  - Choose some initial value for **w**
  - Should we increase or decrease w?
  - Use the slope (gradient)
  - Keep updating **w** by following the gradient, until we reach convergence
  - gradient

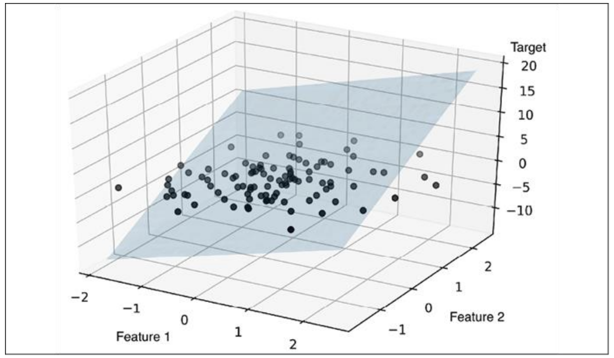
update (w) rule

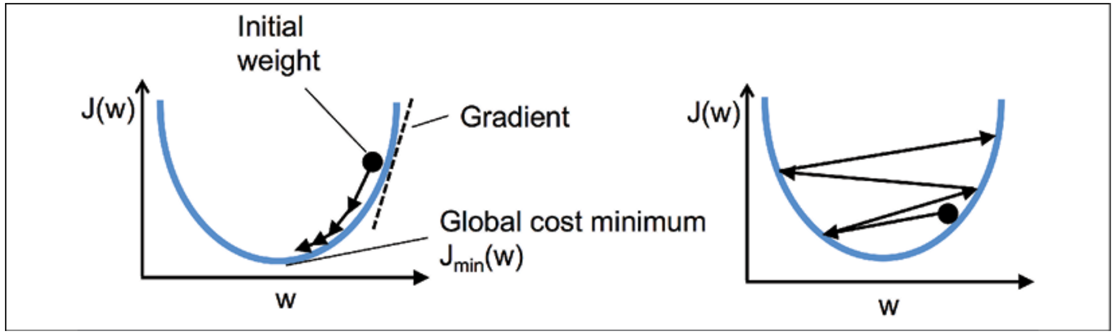
Ex: Linear Regression: MSE

$$\frac{\partial}{\partial w} J \rightarrow \frac{\partial}{\partial w_1} J = \frac{\partial}{\partial w_1} (\hat{y} - y)^2 \rightarrow 2(\hat{y} - y)x$$

$$w := w - \alpha \frac{\partial}{\partial w} J$$



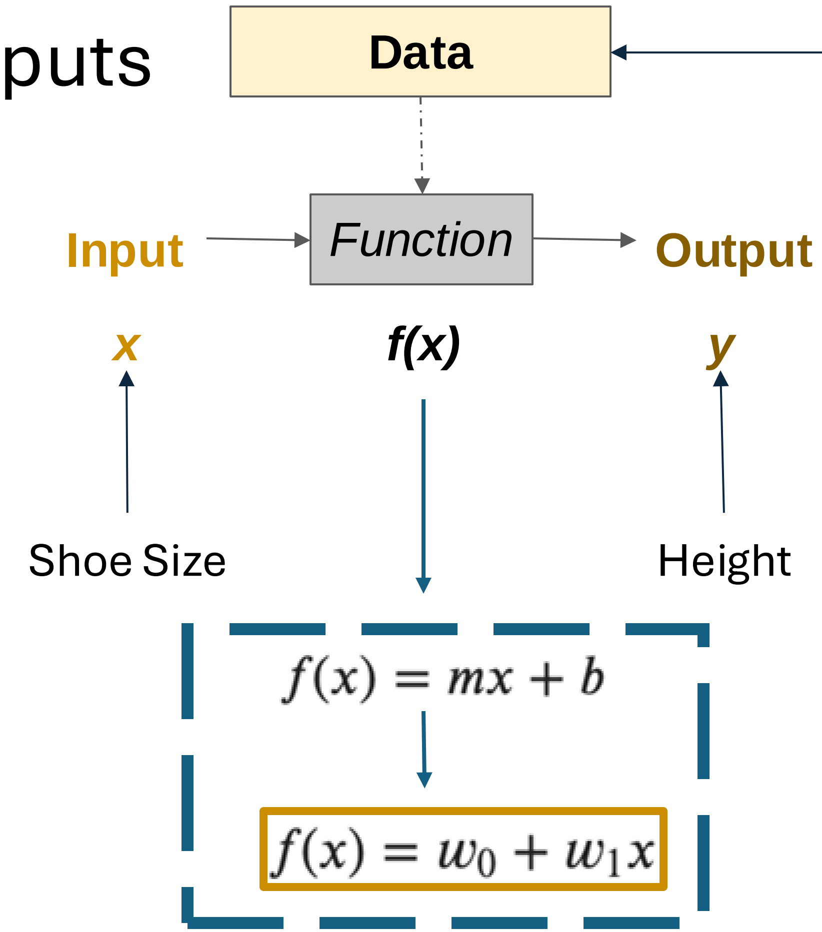


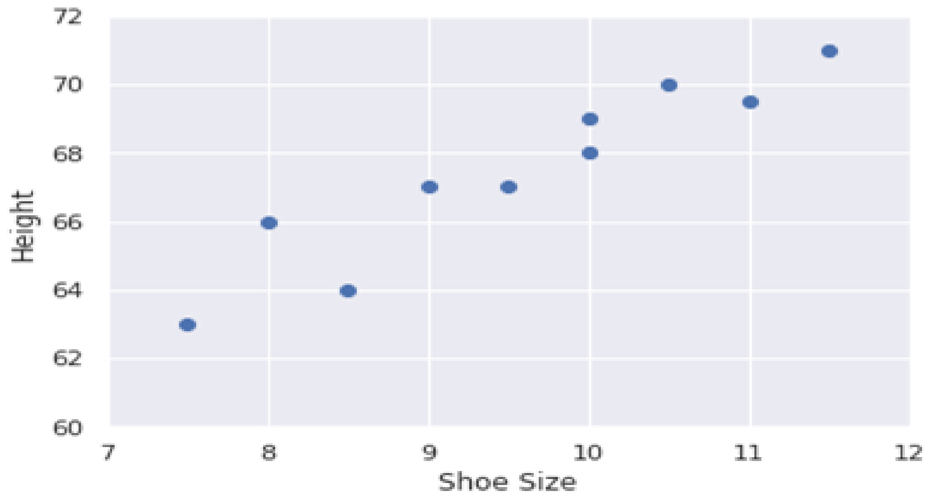


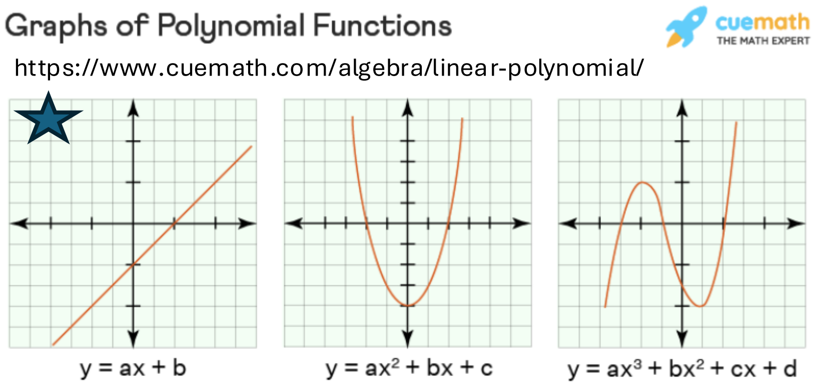
Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

ML Framing: Linear Regression

- Inputs and Outputs
- Labeled data
- Split train/test
- Evaluation
- Baseline
- Use in practice

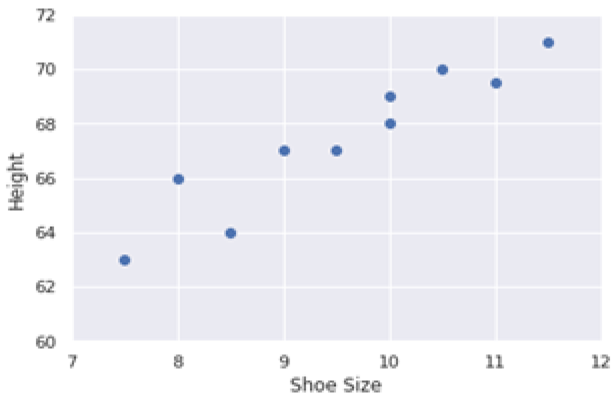






Review: The One-Variable Case

Model $f(x) = w_0 + w_1 x$



Parameters $W = [w_0, w_1]$

Loss $J(w_0, w_1) = \frac{1}{m}\sum_i (w_0 + w_1 x_i - y_i)^2$

Objective $Minimize\ J(w_0, w_1)$

$$MSE = \frac{1}{N}\Sigma_{i=1}^{N}(y_i - \hat{y}_i)^2$$

Update Rule(s) for w

$$w := w - \alpha \frac{\partial}{\partial w} J$$

$$\begin{cases} w_0 := w_0 - \alpha \frac{1}{m}\sum_i (w_0 + w_1 x_i - y_i) \\ w_1 := w_1 - \alpha \frac{1}{m}\sum_i (w_0 + w_1 x_i - y_i) x_i \end{cases}$$

Dot Product & Matrix Transpose

- Sum of the products of the values in **x** and **w** using a (vector) dot product:

$$z = w_0x_0 + w_1x_1 + \cdots + w_mx_m = \sum_{j=0}^{m} x_jw_j = \boldsymbol{w}^T\boldsymbol{x}$$

$$[1 \quad 2 \quad 3] \times \begin{bmatrix} 4 \\ 5 \\ 6 \end{bmatrix} = 1 \times 4 + 2 \times 5 + 3 \times 6 = 32$$

- Transpose (T) is only applicable to matrices
  - In ML:
  - n x 1 or 1 x m matrices are referred to as vectors

- Matrix multiplication ~ vector dot product (YAY NumPy!)
  - Each row in the matrix treated as a single row vector
  - Efficient computation

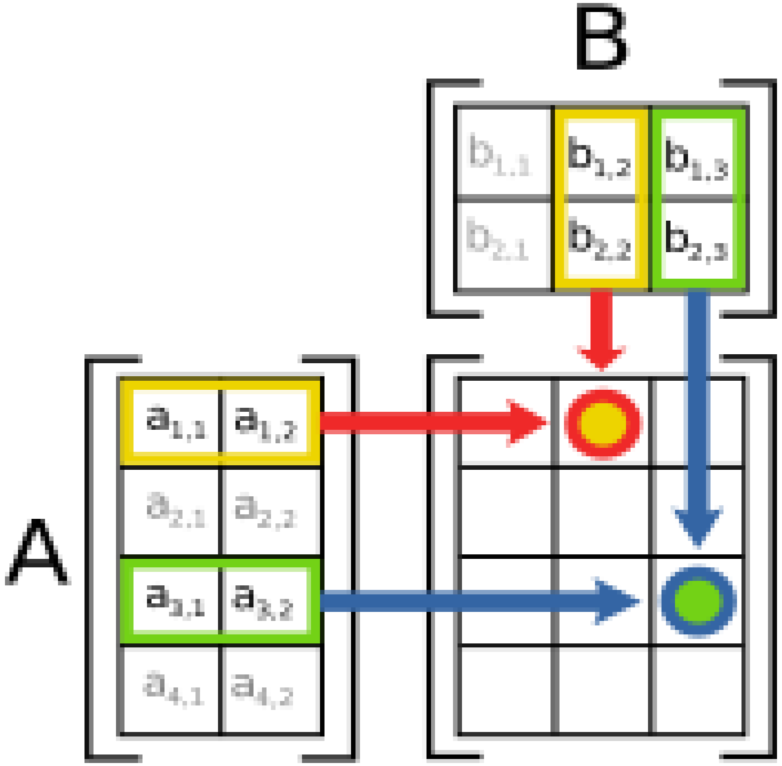
$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \times \begin{bmatrix} 7 \\ 8 \\ 9 \end{bmatrix} = \begin{bmatrix} 1 \times 7 + 2 \times 8 + 3 \times 9 \\ 4 \times 7 + 5 \times 8 + 6 \times 9 \end{bmatrix} = \begin{bmatrix} 50 \\ 122 \end{bmatrix}$$

Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

Matrix Multiplication: Review

$$\underset{4\times 2 \text{ matrix}}{\begin{bmatrix} a_{11} & a_{12} \\ \cdot & \cdot \\ a_{31} & a_{32} \\ \cdot & \cdot \end{bmatrix}} \underset{2\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & b_{12} & b_{13} \\ \cdot & b_{22} & b_{23} \end{bmatrix}} = \underset{4\times 3 \text{ matrix}}{\begin{bmatrix} \cdot & c_{12} & \cdot \\ \cdot & \cdot & \cdot \\ \cdot & \cdot & c_{33} \\ \cdot & \cdot & \cdot \end{bmatrix}}$$

$$c_{12} = a_{11}b_{12} + a_{12}b_{22}$$
$$c_{33} = a_{31}b_{13} + a_{32}b_{23}\,.$$



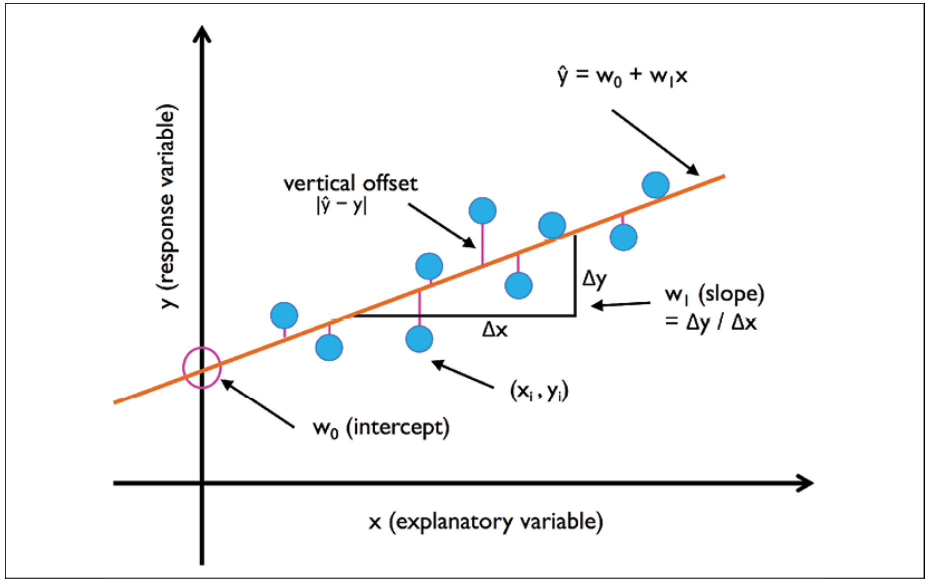
Edited Image Reference: https://en.wikipedia.org/wiki/Matrix_multiplication

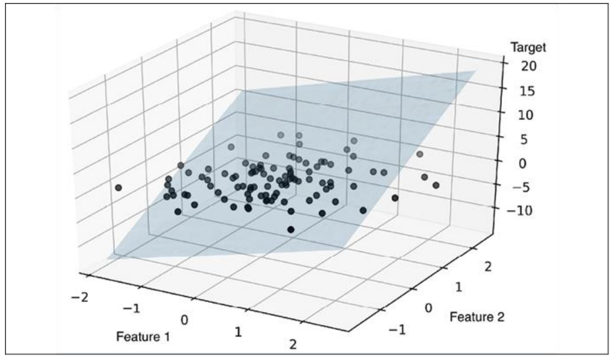
Linear Regression & Gradient Descent (Review)

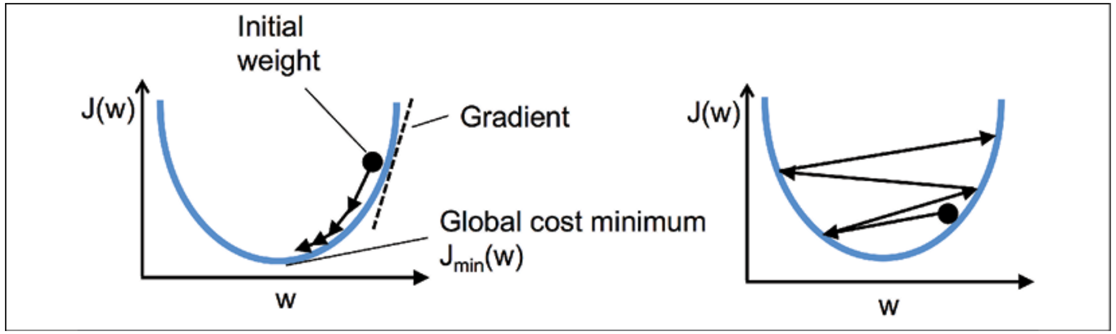
- Training:
  - Get data
  - Initialize weight/s
  - Initialize bias
- Given data
  - Get prediction
  - Get error
  - Update weight, use Gradient Descent
  - Repeat

```
# Run gradient descent
learning_rate = 0.1

preds = np.dot(X, W)
loss = ((preds - Y)**2).mean()
gradient = 2 * np.dot((preds - Y), X) / m
W = W - learning_rate * gradient

print('predictions:', preds)
print('loss:', loss)
print('gradient:', gradient)
print('weights:', W)
```

```
predictions: [6 5]
loss: 16.0
gradient: [ 8. 20. 12.  4.]
weights: [ 0.2 -1.  -0.2  0.6]
```

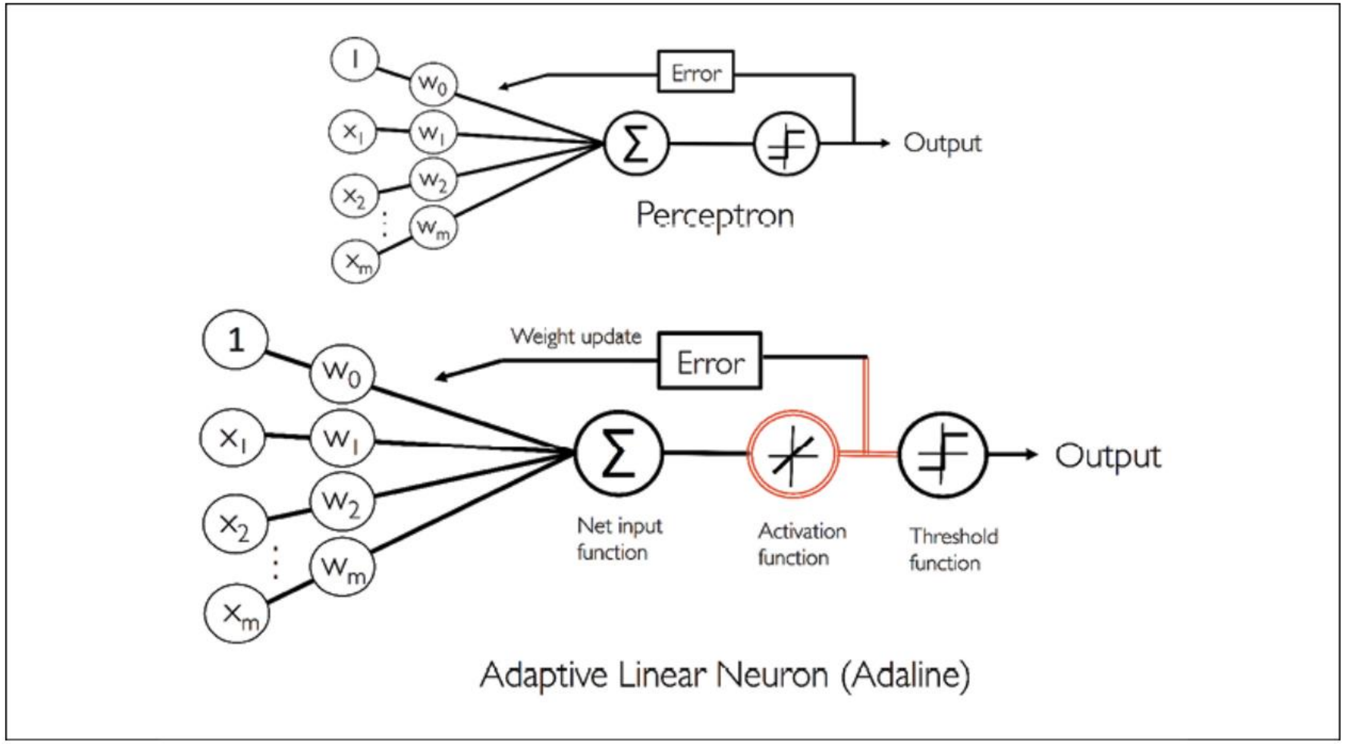






Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

Single Layer Neural Net & Activation Function(s)



Raschka, S., & Mirjalili, V. (2019). Python Machine Learning, Third Edit.

net input $z = \boldsymbol{w}^T\boldsymbol{x}$

$\downarrow$

linear activation function, $\phi(z)$,

$$\phi(\boldsymbol{w}^T\boldsymbol{x}) = \boldsymbol{w}^T\boldsymbol{x}$$

Appendix

Extra slides for review

Closed Form Solution for Solving OLS

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

```
# adding a column vector of "ones"
>>> Xb = np.hstack((np.ones((X.shape[0], 1)), X))
>>> w = np.zeros(X.shape[1])
>>> z = np.linalg.inv(np.dot(Xb.T, Xb))
>>> w = np.dot(z, np.dot(Xb.T, y))
>>> print('Slope: %.3f' % w[1])
Slope: 9.102
>>> print('Intercept: %.3f' % w[0])
Intercept: -34.671
```