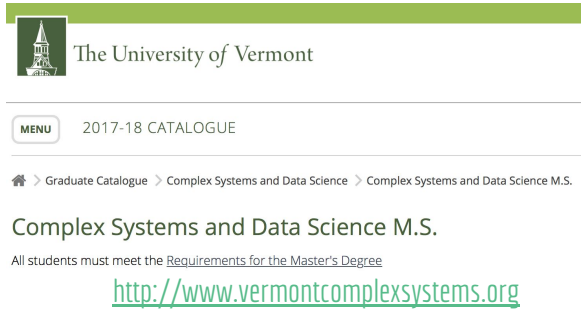
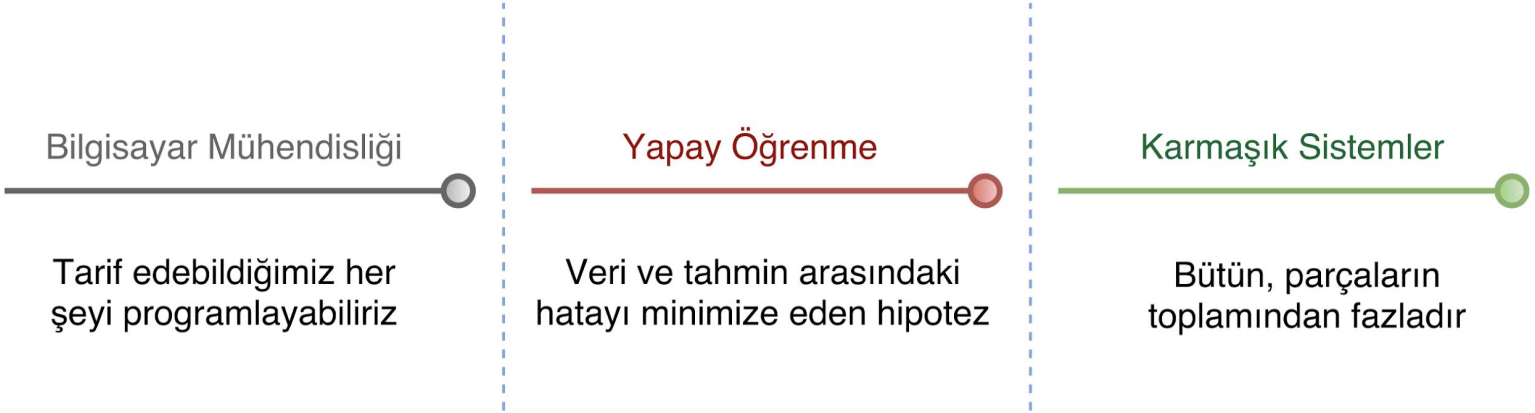
Bilgisayım,
Yapay Zeka ve Karmaşık Sistemler

Dr. Uzay Çetin
Istanbul Bilgi University

https://uzay00.github.io/kahve/orta.html



Bilgisayım, Yapay Zeka, Karmaşık Sistemler



Bilgisayar mühendisliğinin temelinde yatan algoritmik yaklaşıma göre,

- **Tarif edebildiğimiz her şeyi programlayabiliriz**. Peki ya tarif edemediğimiz şeyler?

Evrene Bakış Açısı

Seth Lloyd 2005 yılında basılmış, Programming the Universe adlı kitabında söyle söylüyor,

> Evrendeki her atom, her parçacık bilgi kaydeder. Bu parçacıklar arasındaki her bir çarpışma, ne kadar küçük olursa olsun, meydana gelen her bir değişim sistematik bir biçimde o bilginin işlenmesidir.

Buradaki iki kritik nokta, *bilgi kaydı* ve *bilginin işlenmesidir*. Bilindiği üzere bunlar modern programlamanın temel unsurlarıdır.

> Demek ki, sadece bilgisayarlar bilgi depolayan ve o bilgiyi işleyen aygıtlar değildir. Evrenin kendisi de, baştan sona bilgi işleyen devasa bir sistemdir.

Karmaşık sistemler bilim dalının temellerinde yatan ana fikir,

> **“bütün, parçaların toplamından daha fazladır”** ilkesidir.

Bu parçalarda bulunmayan bir özelliğin, bütünde zuhur edebileceğini ifade eder.

> Örneğin tek başına bir karınca zeki değildir, ama bir bütün halinde koloni en kısa yol problemini çözerbilecek zeka belirtisi gösterir.

Yapay öğrenmenin başarısının arkasında ise “bol miktardaki örneklerden hatayı minimize eden hipotezi bulmak” yatar.

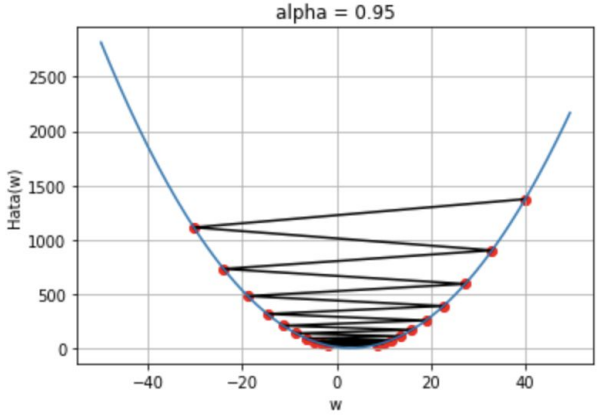
> Bu sayede, elimizdeki örnek veriler ve tahminimiz arasındaki hatayı minimize ederek, algoritmasını yazamadığımız, tarifini bilemediğimiz fakat kolayca yapabildiğimiz yüz tanıma, karakter tanıma gibi işleri bilgisayarlara yaptırabiliriz.

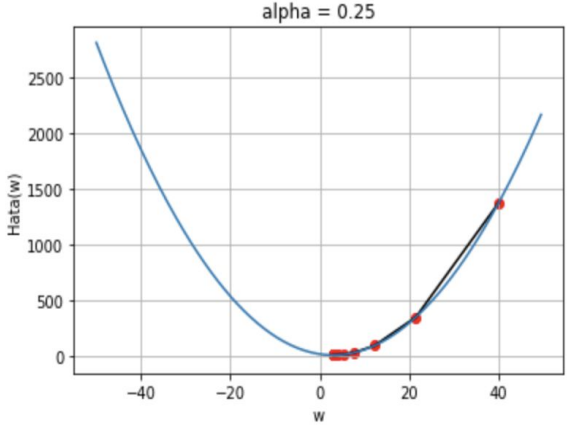
Makine Öğrenmesi, Öğrenmek hatayı minimize eden parametreleri bulmaktır!!

Yapay Öğrenme, bir anlamda optimizasyon problemidir.

```
def Hata(w):
    return ((w-3)**2)+5

def turev(w):
    return 2*(w-3)

def gradyan_inis(w = 0, alpha = 0.05, dongu = 20):
    W = np.zeros(dongu)

    for i in range(dongu):
        W[i] = w
        w = w - alpha * turev(w)

    return W
```

```
a = 0.95
W = gradyan_inis(w = 40, alpha = a)
t = np.arange(-50,50,0.5)

plt.plot(W,Hata(W),'k')
plt.scatter(W,Hata(W), color = 'red')
plt.plot(t,Hata(t))
plt.xlabel('w'); plt.ylabel('Hata(w)'); plt.title("alpha = " + str(a))
plt.grid()
```





El yapımı **“Linear Regression”**

Link:https://drive.google.com/file/d/1t7DwPvKZN1mIoiOOHDLHVG9xifGxrJES/view?usp=sharing

Parameters

$$W = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

Hypothesis

$$h = XW = \begin{bmatrix} x_{00} & x_{01} & x_{02} \\ x_{10} & x_{11} & x_{12} \\ x_{20} & x_{21} & x_{22} \\ x_{30} & x_{31} & x_{32} \\ x_{40} & x_{41} & x_{42} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} 1 & x_{01} & x_{02} \\ 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ 1 & x_{31} & x_{32} \\ 1 & x_{41} & x_{42} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} w_0 + w_1 x_{01} + w_2 x_{02} \\ w_0 + w_1 x_{11} + w_2 x_{12} \\ w_0 + w_1 x_{21} + w_2 x_{22} \\ w_0 + w_1 x_{31} + w_2 x_{32} \\ w_0 + w_1 x_{41} + w_2 x_{42} \end{bmatrix} = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix}$$

Cost Function

$$J(W) = \frac{1}{2m} \sum_{i=1}^{m} (h_i - y_i)^2 = \frac{1}{2m} \sum_{i=1}^{m} ((w_0 + w_1 x_{i1} + w_2 x_{i2}) - y_i)^2$$

Derivative of Cost Funnction

$$\frac{dJ(W)}{dw_j} = \sum_{i=1}^{m} \frac{dJ(W)}{dh_i} \frac{dh_i}{dw_j} = \sum_{i=1}^{m} \frac{1}{m} (h_i - y_i) \frac{dh_i}{dw_j} = \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i) \frac{dh_i}{dw_j} = \frac{1}{m} \sum_{i=1}^{m} (h_i - y_i) x_{ij}$$

Derivative of Cost Funnction

$$\frac{dJ(W)}{dw} = \begin{bmatrix} \frac{dJ(W)}{dw_0} \\ \frac{dJ(W)}{dw_1} \\ \frac{dJ(W)}{dw_2} \end{bmatrix} = \frac{1}{m} \begin{bmatrix} \sum_{i=1}^{m} (h_i - y_i) x_{i0} \\ \sum_{i=1}^{m} (h_i - y_i) x_{i1} \\ \sum_{i=1}^{m} (h_i - y_i) x_{i2} \end{bmatrix}$$

```python
class myRegression():
    def __init__(self,X_train, y_train):
        self.m, self.n = X_train.shape
        self.n += 1 # Add one for x_0 column

        self.X_train = np.hstack((np.ones((self.m,1)), X_train))
        self.y_train = y_train.reshape((self.m,1))
        self.W = np.random.randn(self.n,1)
        print(self.W)

    def cost(self):
        h = self.X_train.dot(self.W)
        return np.sum(np.power(h-self.y_train,2))/ (2*self.m)

    def derivative(self):
        h = self.X_train.dot(self.W)
        derivative = np.sum(self.X_train * (h-self.y_train), axis=0)/ self.m
        return derivative.reshape(self.W.shape)

    def gradient_descent(self, alpha = 0.05, number_steps = 10000):
        for i in range(number_steps):
            self.W = self.W - alpha * self.derivative()
        return self.W

    def predict(self, X_test):
        m, n = X_test.shape
        X_test = np.hstack((np.ones((m,1)), X_test))
        return X_test.dot(self.W)

    def fit(self):
        self.W = self.gradient_descent()
```

El yapımı **“Logistic Regression”**

Link:https://drive.google.com/file/d/1t7DwPvKZN1mIoiOOHDLHVG9xifGxrJES/view?usp=sharing

Sigmoid Function

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

Parameters

$$W = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}$$

Hypothesis

$$h = g(XW) = g(\begin{bmatrix} 1 & x_{01} & x_{02} \\ 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ 1 & x_{31} & x_{32} \\ 1 & x_{41} & x_{42} \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix}) = \begin{bmatrix} g(w_0 + w_1 x_{01} + w_2 x_{02}) \\ g(w_0 + w_1 x_{11} + w_2 x_{12}) \\ g(w_0 + w_1 x_{21} + w_2 x_{22}) \\ g(w_0 + w_1 x_{31} + w_2 x_{32}) \\ g(w_0 + w_1 x_{41} + w_2 x_{42}) \end{bmatrix} = \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \\ h_4 \end{bmatrix}$$

Cost Function

$$J(W) = \frac{1}{2m}\sum_{i=1}^{m}(h_i - y_i)^2 = \frac{1}{2m}\sum_{i=1}^{m}(g(w_0 + w_1 x_{i1} + w_2 x_{i2}) - y_i)^2$$

Derivative of Cost Funnction

$$\frac{dJ(W)}{dw_j} = \sum_{i=1}^{m}\frac{dJ(W)}{dh_i}\frac{dh_i}{dz_i}\frac{dz_i}{dw_j} = \sum_{i=1}^{m}\frac{1}{m}(h_i - y_i)\frac{dh_i}{dz_i}\frac{dz_i}{dw_j} = \frac{1}{m}\sum_{i=1}^{m}(h_i - y_i)h_i(1-h_i)\frac{dz_i}{dw_j} = \frac{1}{m}\sum_{i=1}^{m}(h_i - y_i)h_i(1-h_i)x_{ij}$$

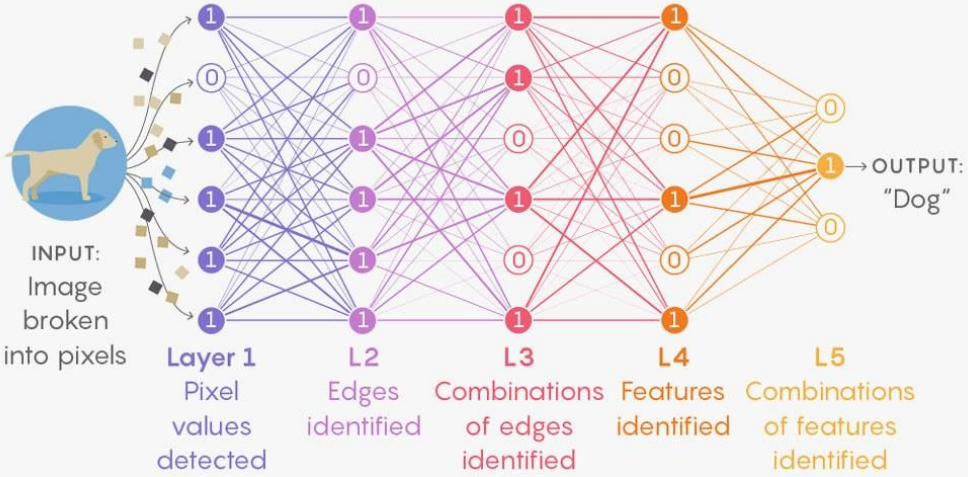
Derivative of Cost Funnction

$$\frac{dJ(W)}{dw} = \begin{bmatrix} \frac{dJ(W)}{dw_0} \\ \frac{dJ(W)}{dw_1} \\ \frac{dJ(W)}{dw_2} \end{bmatrix} = \frac{1}{m}\begin{bmatrix} \sum_{i=1}^{m}(h_i - y_i)h_i(1-h_i)x_{i0} \\ \sum_{i=1}^{m}(h_i - y_i)h_i(1-h_i)x_{i1} \\ \sum_{i=1}^{m}(h_i - y_i)h_i(1-h_i)x_{i2} \end{bmatrix}$$

```python
def sigmoid(z):
    return(1 / (1 + np.exp(-z)))

class myClassification():
    def __init__(self,X_train, y_train):
        self.m, self.n = X_train.shape
        self.n += 1 # Add one for x_0 column

        self.X_train = np.hstack((np.ones((self.m,1)), X_train))
        self.y_train = y_train.reshape((self.m,1))
        self.W = np.random.randn(self.n,1)

    def cost(self):
        h = sigmoid(self.X_train.dot(self.W))
        return np.sum(np.power(h-self.y_train,2))/ (2*self.m)

    def derivative(self):
        h = sigmoid(self.X_train.dot(self.W))
        derivative = np.sum(self.X_train * (h-self.y_train) * h * (1-h) , axis=0)/ self.m
        return derivative.reshape(self.W.shape)

    def gradient_descent(self, alpha = 0.05, number_steps = 10000):
        for i in range(number_steps):
            self.W = self.W - alpha * self.derivative()
        return self.W

    def predict(self, X_test, threshold=0.5):
        m, n = X_test.shape
        X_test = np.hstack((np.ones((m,1)), X_test))
        h = sigmoid(X_test.dot(self.W))
        p = h >= threshold
        return (p.astype('int'))

    def fit(self):
        self.W = self.gradient_descent()
```

Learning From Experience

Deep neural networks learn by adjusting the strengths of their connections to better convey input signals through multiple layers to neurons associated with the right general concepts.

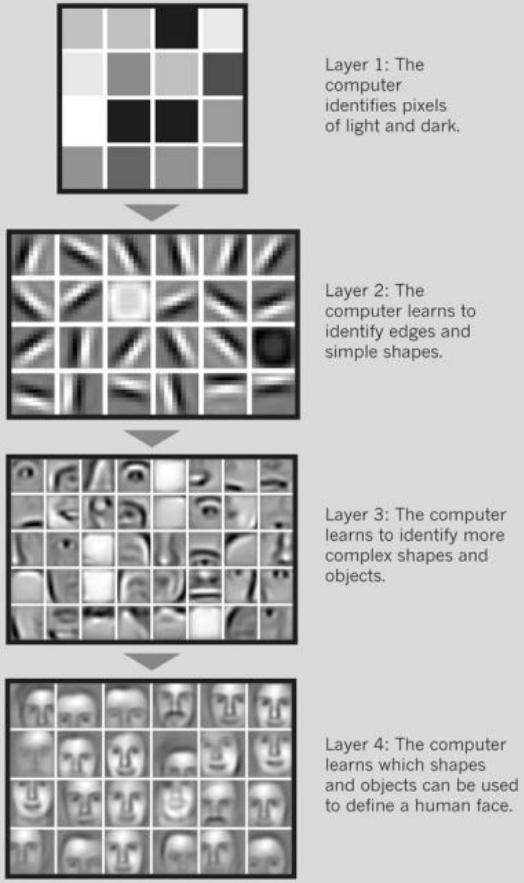


When data is fed into a network, each artificial neuron that fires (labeled "1") transmits signals to certain neurons in the next layer, which are likely to fire if multiple signals are received. The process filters out noise and retains only the most relevant features.

Source: LUCY READING-IKKANDA/QUANTA MAGAZINE

FACIAL RECOGNITION

Deep-learning neural networks use layers of increasingly complex rules to categorize complicated shapes such as faces.



Source: http://www.cifasis-conicet.gov.ar/granitto/ECI2014/Jones14.pdf