

רגרסיה לוגית

perceptron

גלעד מרקמן

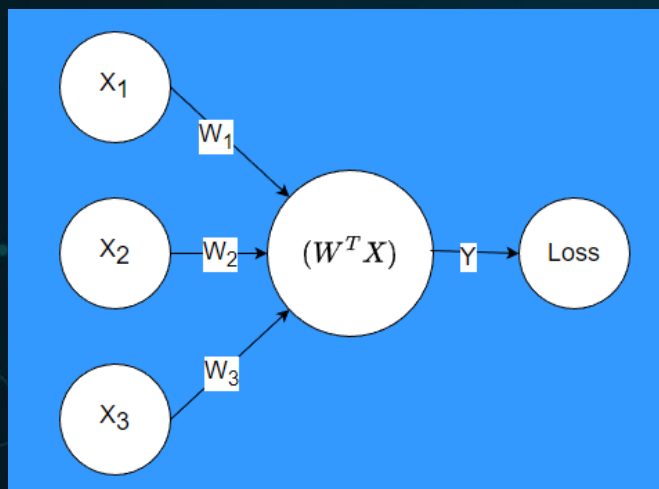


קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות



גרף רגרסיה לינארית

- המודל שבנינו עד כה היה מודל לינארי שכלל רק פעולות אלגבריות של כפל וחיבור.



- נהוג לשרטט את המודל הלינארי כגרף הכולל:

- צמתים לקבלת מידע (input) בשרטוט הם $X_1..X_3$.

- משקלים W

- צומת אשר מבצע את החישוב הלינארי:

$$W^T X = X_1 * W_1 + X_2 * W_2 + X_3 * W_3$$

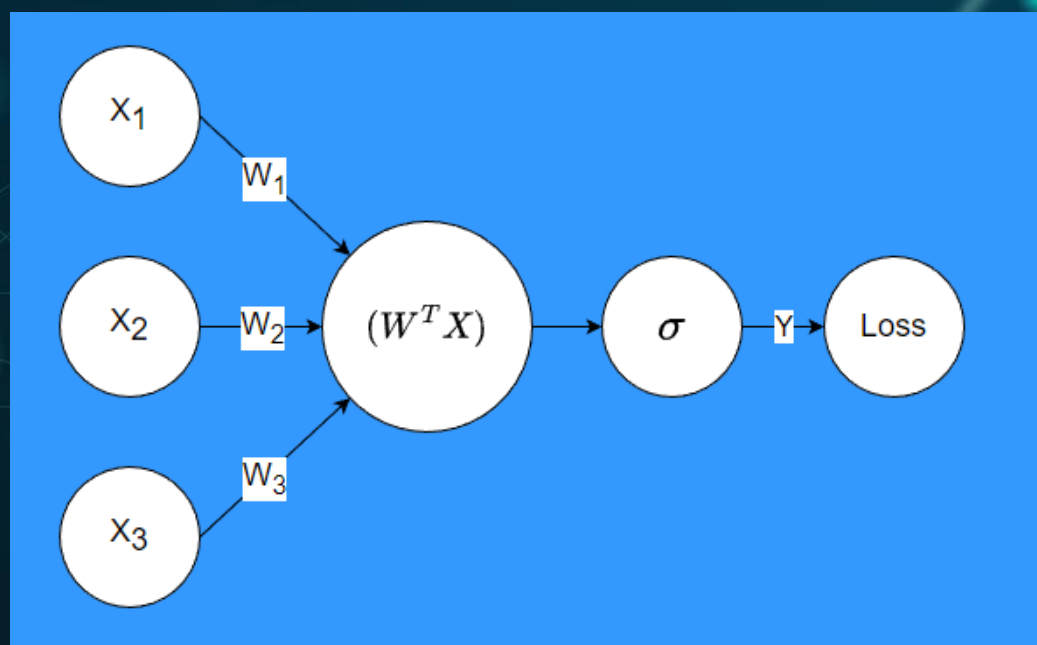
- חישוב המחיר Loss/Cost.

- בעזרת רגרסיה לינארית עדכנו את המשקלים באמצעות חלחול לאחור.



perceptron

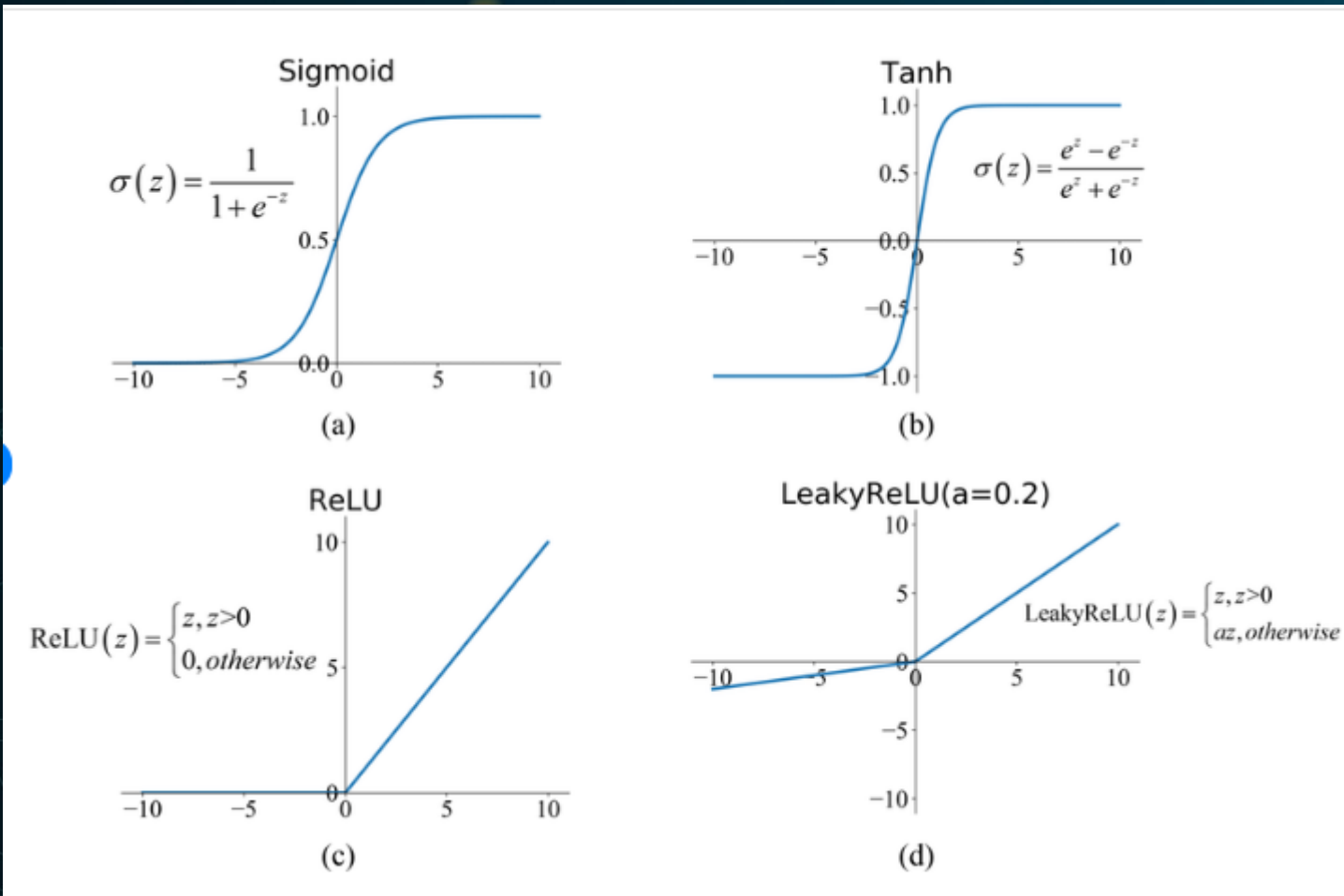
- הרגרסיה הלינארית מוגבלת רק לבעיות שהפתרון הוא לינארי. אולם, במרבית הבעיות שבהן אנו נתקלים המודל הלינארי אינו מספק.
- על כן, פותח ה perceptron ש"שובר" את הלינאריות כל ידי הוספת פונקציה לא לינארית המכונה פונקציית אקטיבציה, המסומנת באות σ .



פונקציות אקטיבציה



קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

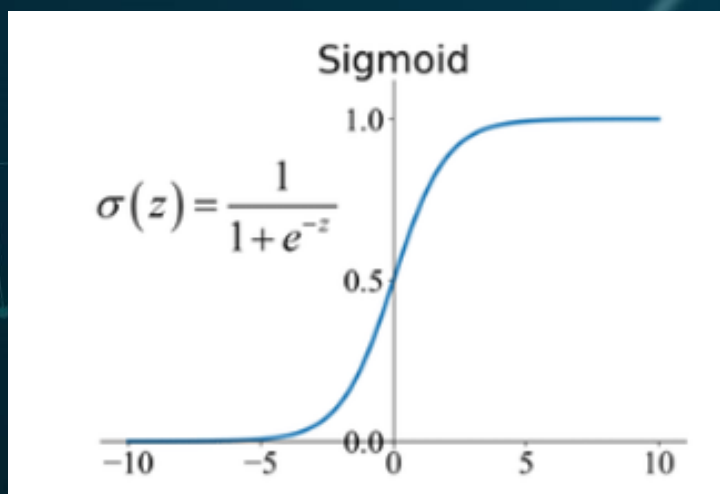


- ישנן מספר פונקציות אקטיבציה המקובלות בלמידת מכונה.
- נביא את הנפוצות שביניהן.
- בנוסף קיימת פונקציה חשובה Softmax בה נדון בשיעור הבא.
- בחירת פונקציית האקטיבציה אף היא נעשית בדרך של ניסוי וטעיה.



Sigmoid

- פונקציית ה Sigmoid מחזירה לנו ערכים בין 0 ל - 1.
- אנו משתמשים בפונקציה זו כאשר התשובה שאנו מחפשים היא בינארית (חיובי או שלילי).
- אם התוצאה גדולה מ 0.5 נחשיב זאת כחיובי - 1.
- אם התוצאה קטנה מ 0.5 נחשיב זאת כשלילי - 0.





פונקציית המחיר loss / cost

- פונקציית המחיר משתנה בהתאם לפונקציית האקטיבציה בה נשתמש בשכבה האחרונה – שכבת הפלט.
- להלן טבלת פונקציות האקטיבציה עם פונקציות המחיר המתאימות להן.

Activation Function	Cost Function
Sigmoid	Binary Cross Entropy
Softmax	Categorical Cross Entropy
Rectified Linear Unit (ReLU)	Mean Squared Error
Hyperbolic Tangent (Tanh)	Binary Cross Entropy, Mean Squared Error
Leaky ReLU	Mean Squared Error

- הטבלה הוכנה בעזרת chatGPT-4
- הנוסחאות המתמטיות לכל אחת מפונקציות המחיר אני משאיר ללמידה עצמית.



טבלת פונקציות PyTorch

- ספריית פיתון כוללת את כל פונקציות האקטיבציה ופונקציות המחיר המתאימות להן, ואנו מוסיפים אותן למודל שלנו בהתאם לצורך.

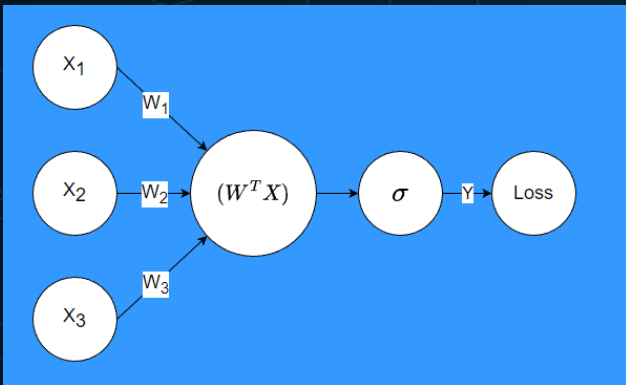
Activation Function	Cost Function
<code>torch.nn.Sigmoid()</code>	<code>torch.nn.BCELoss()</code>
<code>torch.nn.Softmax()</code>	<code>torch.nn.CrossEntropyLoss()</code>
<code>torch.nn.ReLU()</code>	<code>torch.nn.MSELoss()</code>
<code>torch.nn.Tanh()</code>	<code>torch.nn.BCELoss()</code> , <code>torch.nn.MSELoss()</code>
<code>torch.nn.LeakyReLU()</code>	<code>torch.nn.MSELoss()</code>

- הטבלה הוכנה בעזרת chatGPT-4

Perceptron in PyTorch

```
# design model
Model = nn.Sequential(
    nn.Linear(30,1),
    nn.Sigmoid()
)

#construct loss and optimizer
Loss = nn.BCELoss()
```



- בניית ה Perceptron נעשית על ידי הוספת פונקציית אקטיבציה לאחר הפונקציה הלינארית.
- המחלקה nn.Sequential (סדרתי) בונה לנו מודל המבצע את החישובים לפי השכבות שנקבעו:
- תחילה יבוצע חישוב לינארי.
- על התוצאה תופעל פונקציית האקטיבציה (במקרה זה sigmoid).
- ניתן להוסיף בדרך דומה שכבות נוספות (ראה בהמשך).
- פונקציית המחיר (הטעות) – יש להתאים את הפונקציה המתאימה לפונקציית האקטיבציה בשכבה הראשונה.

הלכה למעשה Breast Cancer Dataset



קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

- נדגים את השימוש בפונקציות אקטיבציה באמצעות בסיס הנתונים breast_cancer_dataset של בית החולים בויסקונסין, ארה"ב.
- בסיס הנתונים כולל 569 תוצאות בדיקות הדמיה של גושים שנחשדו כסרטן השד. כל דגימה כוללת 30 פרמטרים, כגון: רדיוס, מרקם, סימטריות וכד'. לכל דוגמה ניתנת התשובה לאחר שבוצעה ביופסיה (ניתוח) והתברר אם המדובר בגוש סרטני או שפיר (תשובה בינארית).
- אנחנו מתבקשים לבנות מודל שיעריך את סוג הסרטן בהתאם לפרמטרים הנתונים, כך שניתן להעריך אם הגוש סרטני מייד לאחר הבדיקה וללא צורך בניתוח.



הכנת הנתונים

- את הנתונים ניתן לטעון באמצעות ספרייה `sklearn`, אשר כוללת מספר בסיסי נתונים לתרגול למידת מכונה.
- טעינת בסיסית הנתונים נעשית באמצעות המחלקה `datasets`.
- הפעולה `train_test_split` מאפשרת לנו לפצל את הדאטה לשתי קבוצות (פיצול אקראי), בהתאם ליחס שנבחר:
 - `train` – לאימון המודל – בחרנו ב- 80% מהדאטה
 - `test` – לבדיקת המודל – בחרנו ב- 20% מהדאטה.

```
import torch
import torch.nn as nn
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.model_selection import train_test_split
```



הכנת הנתונים – טעינה וחלוקה לקבוצות

- טעינת הנתונים מתבצעת למערכי `numpy`.
- ניתן לראות כי y מקבל שני ערכים: גידול שפיר וממאיר.
- מערך X של הנתונים כולל 569 דגימות (שורות) שכל אחת כולל 30 פרמטרים שונים.
- מערך Y של התוצאות כולל 569 תוצאות עם ערך בודד (בוליאני)

```
bc = datasets.load_breast_cancer()
X, y = bc.data, bc.target
print ("target_names",bc.target_names)
print ("X, y",X.shape, y.shape)
```

```
target_names ['malignant' 'benign']
X, y (569, 30) (569,)
```



רגרסיה לינארית על טבלה

- נשים לב שאנחנו מכניסים למודל הלינארי) בבת אחת 455 דוגמאות, שכל אחת מהם כוללת 30 פרמטרים.
- המודל שלנו יודע לבצע חישוב על טבלה בה כל שורה מהווה דוגמה. אין צורך לבצע לולאה על כל דוגמה.

```
n_samples, n_features = X.shape
X_train_np, X_test_np, y_train_np, y_test_np = train_test_split(X,y, test_size=0.2, random_state = 2)

X_train = torch.from_numpy(X_train_np.astype(np.float32))
X_test = torch.from_numpy(X_test_np.astype(np.float32))

y_train = torch.from_numpy(y_train_np.astype(np.float32))
y_test = torch.from_numpy(y_test_np.astype(np.float32))

y_train = y_train.view(y_train.shape[0],1)
y_test = y_test.view(y_test.shape[0],1)

print (X_train.shape, y_train.shape)
print (X_test.shape, y_test.shape)

torch.Size([455, 30]) torch.Size([455, 1])
torch.Size([114, 30]) torch.Size([114, 1])
```

פיצול הנתונים
למערכי numpy

הכנת הנתונים – נרמול



קריית החינוך
פארק המדע
בית לערכים
למציאות ולחדשנות

```
max, __ = X_train.max(dim=0)
min, __ = X_train.min(dim=0)
def Normalize_minMax(X, max, min):
    max = max.expand(X.shape[0], -1)
    min = min.expand(X.shape[0], -1)
    return (X - min) / (max - min)
```

```
mean = X_test.mean(dim=0)
std = X_test.std(dim=0)
def Normalize_z(X, mean, std):
    return (X - mean) / std
```

```
print (max.shape, min.shape)
print (mean.shape, std.shape)
```

```
torch.Size([30]) torch.Size([30])
torch.Size([30]) torch.Size([30])
```

```
def Normalize_minMax(X, max, min):
    max = max.expand(X.shape[0], -1)
    min = min.expand(X.shape[0], -1)
    return (X - min) / (max - min)
```

```
def Normalize_z(X, mean, std):
    return (X - mean) / std
```

```
# X_train = Normalize_minMax(X_train, max, min)
# X_test = Normalize_minMax(X_test, max, min)
X_train = Normalize_z(X_train, mean, std)
X_test = Normalize_z(X_test, mean, std)
```



קביעת פרמטרים, בניית המודל ופונקציית ההפסד

- השתמשנו במחלקה nn.Sequential המאפשרת לנו לבנות מודל של רשת נוירונים. הפרמטרים המועברים לבנאי המחלקה הינם השכבות.
- שכבה ראשונית לינארית ולאחריה פונקציית אקטיבציה.

```
learning_rate = 0.1
epochs = 500
losses = torch.zeros(epochs) # tensor to save losses for print

# design model
Model = nn.Sequential(
    nn.Linear(30,1),
    nn.Sigmoid()
)

#construct loss and optimizer
Loss = nn.BCELoss()

# init optimizer
optim = torch.optim.Adam(Model.parameters(), lr=learning_rate)
```

בחירה בפונקציה
Sigmoid כיוון
שהתוצאה שלנו היא
בינארית: ממאיר או
שפיר

לולאת האימון לא משתנה

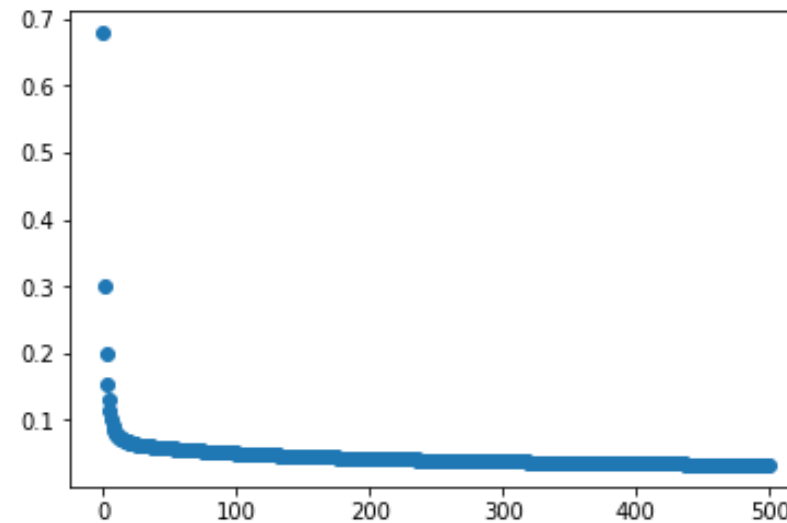


קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

```
for epoch in range(epochs):  
    # forward  
    y_predict = Model(X_train)  
  
    # backward  
    loss = Loss(y_predict, y_train)  
    loss.backward()  
  
    # update wights  
    optim.step()  
    losses[epoch]=loss  
  
    if epoch % 10 == 0:  
        print(f"epoch= {epoch} loss={loss.item():.4f} ")  
  
    # zero wights  
    optim.zero_grad()
```

```
plt.plot(losses.detach(), 'o')  
plt.show
```

```
<function matplotlib.pyplot.show(close=None, block=None)>
```



בדיקת המודל על נתוני הניסוי X-test



קריית החינוך
פארק המדע
בית לערכים
למצוינות ולחדשנות

• הגענו לדיוק של למעלה מ-95%.

```
with torch.no_grad():  
    y_predicted = Model(X_test)  
    y_predicted_bin = y_predicted.round()  
    accuracy = y_predicted_bin.eq(y_test).sum() / float(y_test.shape[0])  
    print (f'accuracy = {accuracy:.4f}')
```

```
print(y_test.shape)  
print(y_predicted_bin.shape)
```

```
accuracy = 0.9561  
torch.Size([114, 1])  
torch.Size([114, 1])
```

```
print ("predicted, test \n", torch.cat((y_predicted_bin, y_test), dim=1))
```

```
[1., 1.],  
[1., 1.],  
[0., 0.],  
[1., 1.],  
[1., 1.],  
[0., 0.],  
[1., 1.],  
[0., 0.],  
[0., 0.],  
[1., 1.],  
[0., 0.],  
[0., 0.],  
[0., 1.],  
[0., 0.],  
[0., 0.],  
[0., 0.],  
[1., 1.]
```




קריית החינוך
פארק המדע
בית לערכים
למצינות ולחדשנות

סיכום

- רגרסיה לוגית נעשית בדיוק כמו רגרסיה לינארית. ההבדל היחידי הוא שלאחר החישוב הלינארי אנו מוסיפה פונקציית אקטיבציה מתאימה.
- בנוסף עלינו לשנות את פונקציית המחיר שתתאים לפונקציית האקטיבציה.
- למעשה בנינו רשת נוירונים הכוללת שכבה אחת. בשיעור הבא נבנה רשת נוירונים הכוללת מספר שכבות.