# Working with tensors

PyTorch

- שלבי עבודה ב Py torch
- אופטימייזרים
- פונקציות אקטיבציה

## שלבי עבודה עם מודלים בpy torch

1. **הגדרת המודל**: יצירת מחלקה (שיורשת מ nn.Model בד"כ)

- עם **שכבות נוירונים**
- עם **פונקציית Forward**

```python
# Define Neural Network Architecture
class Moon(nn.Module): # almost all PyTorch models are subclasses of nn.Module
  def __init__(self):
    super().__init__()
    # nn.Linear fc layers handle the input and output shapes
    self.fc1 = nn.Linear(in_features=2,out_features=16)#fc=fully connected layer
    self.relu1 = nn.ReLU()
    self.fc2 = nn.Linear(in_features=16, out_features=8)
    self.relu2 = nn.ReLU()
    self.fc3 = nn.Linear(in_features=8, out_features=1)

  # Define the forward method
  def forward(self, x): # No sigmoid here, BCEWithLogitsLoss will handle that
    x = self.fc1(x) # Pass input through first fully connected layer
    x = self.relu1(x)
    x = self.fc2(x)
    x=self.relu2(x)
    x = self.fc3(x) # Pass input through second fully connected layer
    return x
```

## שלבי עבודה עם מודלים בpy torch

3. יצירת מופע/ **אינסטנס** – יצירת אובייקט מהמודל שהוגדר.
4. הגדרת פונקציית עלות
5. הגדרת **אופטימייזר** (ממטב)- בחירת הדרך בה יעודכנו המשקלים
6. **יצירת לולאה אימון** הכוללת:

חיזוי,

חישוב עלות (הפסד\Loss) ,

חישוב שיפועים (גרידאנטים)

ועדכון משקלים בכל איטרציה

```
# Instantiate Model and Select Loss Function/Optimizer
model = Moon()
criterion = nn.BCEWithLogitsLoss()
optimizer = optim.SGD(model.parameters(), lr=0.1)
```

```
# Training Loop
epochs = 1000
for epoch in range(epochs):
  model.train() #training mode
  optimizer.zero_grad()
  outputs = model(X_train_tensor)
  loss = criterion(outputs, y_train_tensor)
  loss.backward()
  optimizer.step()
```

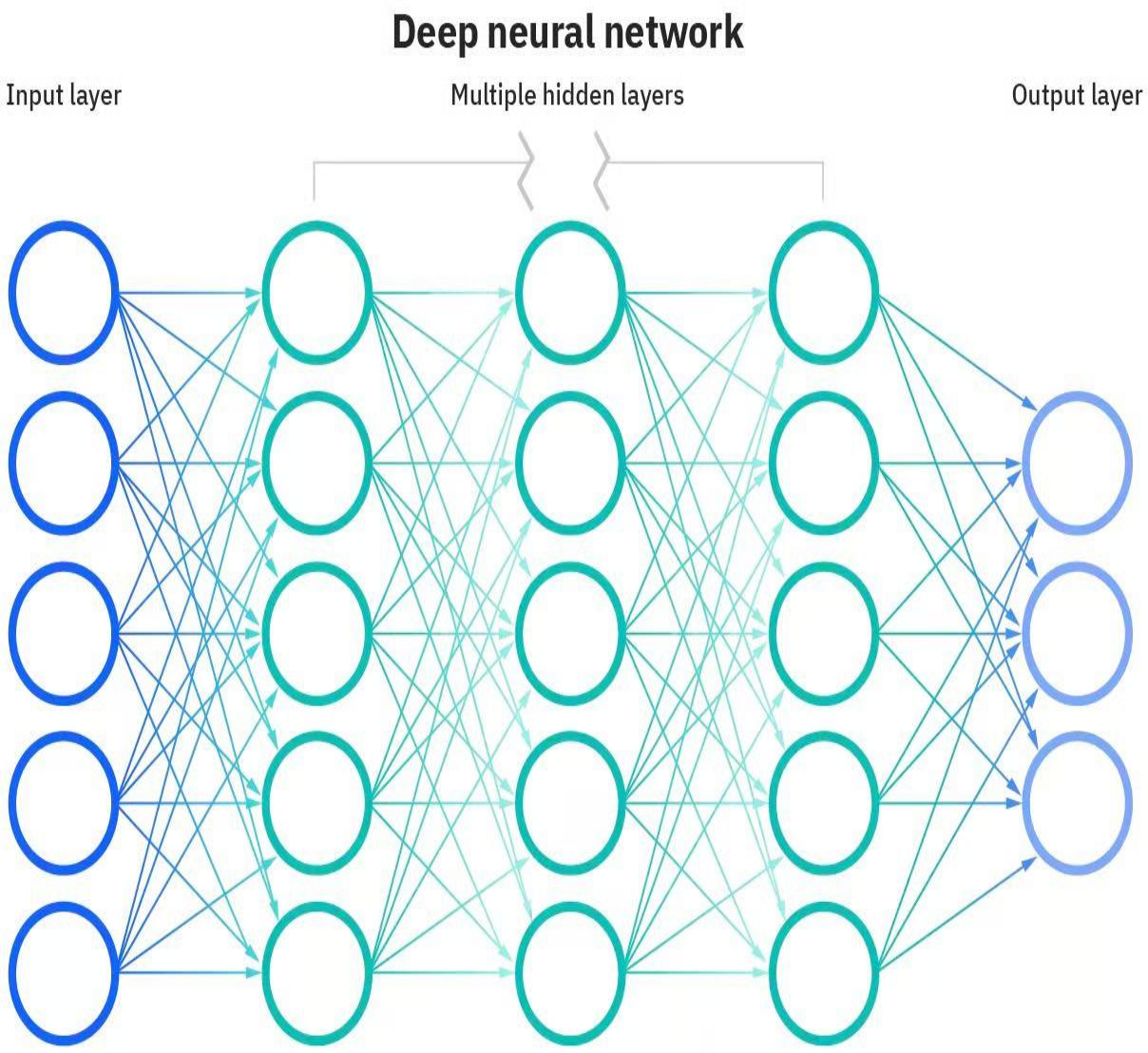
## שלבי עבודה עם מודלים בpy torch

7. הערכה- העברת המודל למשב בדיקה על מנת לבדוק את טיבו, הפעלת המודל לחיזוי ללא מעקב שיפועים.

```
model.eval() # Evaluation Mode On!
with torch.no_grad():
  predictions  = torch.sigmoid(model(X_test_tensor))
  predicted = (predictions > 0.5).float()
  accuracy = (predicted.eq(y_test_tensor).float()).mean()
  print(f"Accuracy: {accuracy.item()}")
```

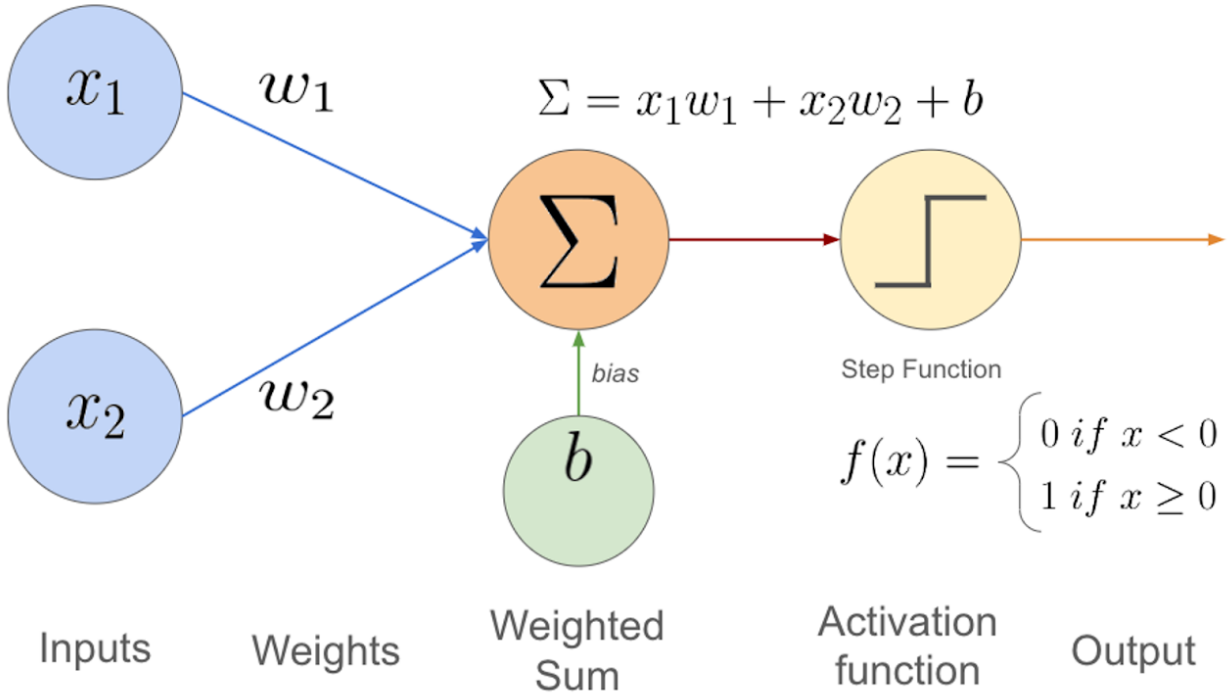
| Loss Function | Use Case | PyTorch |
|---|---|---|
| BCELoss | Binary | nn.BCELoss |
| BCEWithLogitsLoss | Binary | nn.BCEWithLogitsLoss |
| CrossEntropyLoss | Multi-class | nn.CrossEntropyLoss |
| MAELoss or L1Loss | Regression | nn.L1Loss |
| MSELoss or L2Loss | Regression | nn.MSELoss |

| Optimizer | PyTorch | יתרון | חיסרון |
|---|---|---|---|
| SGD | optim.SGD | פשוט ומייצב למידה | עלול להיות איטי ומתקשה למצוא מינימום גלובלי בבעיות מורכבות |
| Adam | optim.Adam | יעיל ברשתות עמוקות, מתמודד טוב עם נתונים משתנים ולא יציבים | עלול להתכנס מהר מדי לפתרון מקומי ולא להגיע לאופטימום גלובלי |
| AdaGrad | optim.Adagrad | מתאים לבעיות עם תכונות נדירות ( sparse features) | קצב הלמידה קטן מדי לאורך הזמן, מה שגורם לעצירת למידה |
| Momentum | optim.SGD (with momentum) | משפר את קצב הלמידה על ידי שמירה על מומנטום מהעבר | רגיש לכוונון של הפרמטרים (למשל גודל המומנטום) |



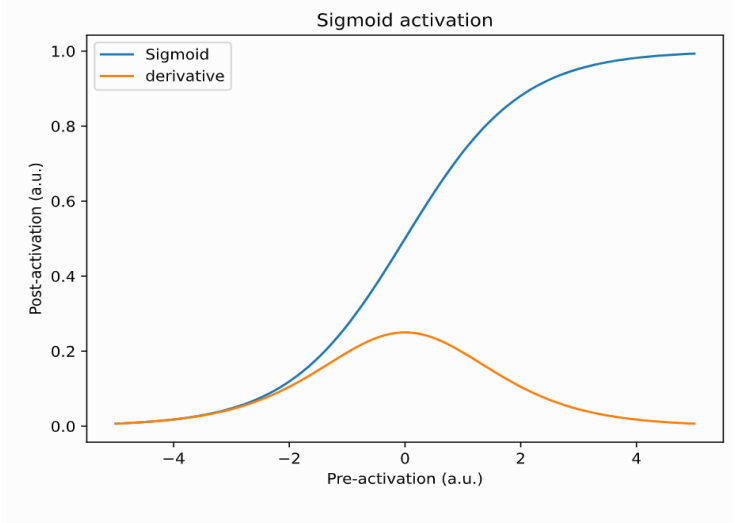

בכל פרספטרון יחיד מחושבים המשקלים וההטיה
ואז פונקציית האקטיבציה היא זו שעל פיה יוחלט : אם, ומה ירה הפרספטרון

ברשת של נוירונים פונקציית האקטיבציה עובדת על כל הנוירונים בשכבה.



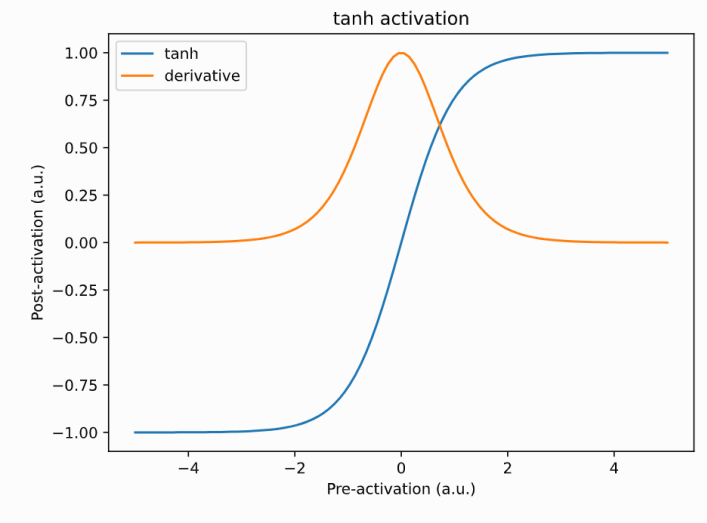

# Activation Functions

## Sigmoid Function



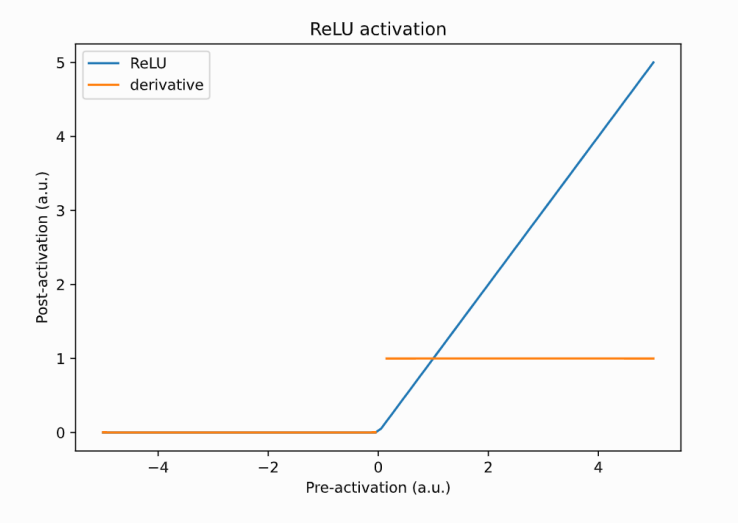

$$S(x) = \frac{1}{1 + e^{-x}}$$

## Hyperbolic Tangent




$$f(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

## Rectified Linear Unit (Relu)




$$f(x) = max\ (0, x)$$

- אין פתרון "אחד שמתאים לכולם". התאמת פונקציות אקטיבציה, פונקציות הפסד ואופטימייזרים תלויה בבעיה הספציפית.
- שימו לב לבחירת הפונקציות בהתבסס על המבנה והדרישות של הנתונים.
- המשיכו להתנסות ולבחון תוצאות עם נתונים שונים כדי להבין כיצד אקטיבציות ואופטימייזרים משפיעים על התוצאה הסופית .