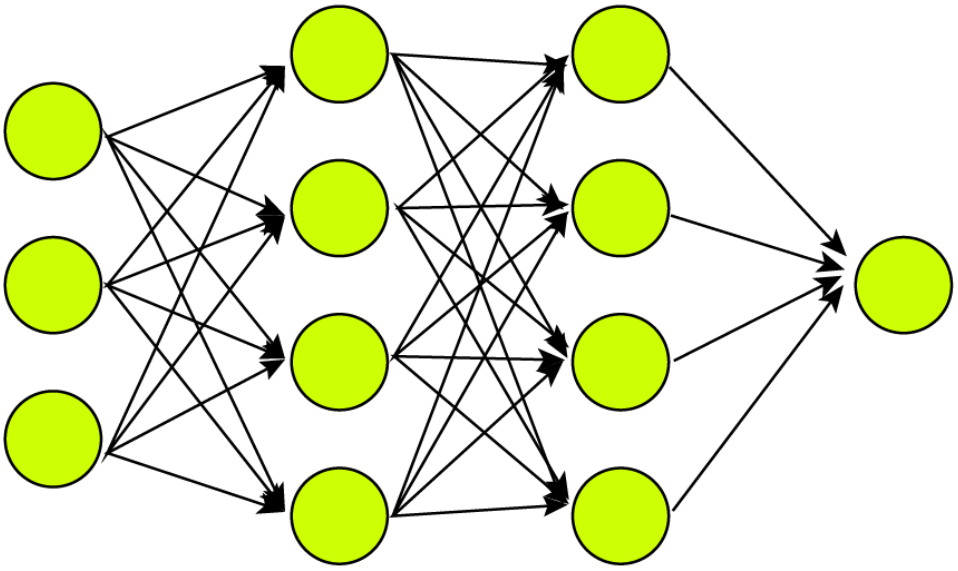
შეხვედრა 12: შესავალი ნეირონულ ქსელებსა და ღრმა სწავლებაში (Deep Learning)

წინა შეხვედრებზე ჩვენ გავეცანით მანქანური სწავლების საფუძვლებს და ავაგეთ ჩვენი პირველი კლასიფიკაციის ვექტორული მოდელი. დღეს კი დროა, შევისწავლოთ ხელოვნური ინტელექტის ყველაზე ძლიერი და ინოვაციური კონცეფცია - **ნეირონული ქსელები** და **ღრმა სწავლება**. ნეირონული ქსელები არის კომპიუტერული მოდელები, რომლებიც ადამიანის ტვინის სტრუქტურითაა შთაგონებული. ისინი შედგება ერთმანეთთან დაკავშირებული ხელოვნური „ნეირონებისგან“, რომლებიც ინფორმაციას ამუშავებენ. ამ შეხვედრის ბოლოს შენ შეძლებ გაიგო, თუ როგორ მუშაობს ნეირონული ქსელის მარტივი სტრუქტურა და დაწერო პროგრამა, რომელიც აღწერს მის ძირითად კომპონენტებს.

1. რა არის ნეირონული ქსელი?

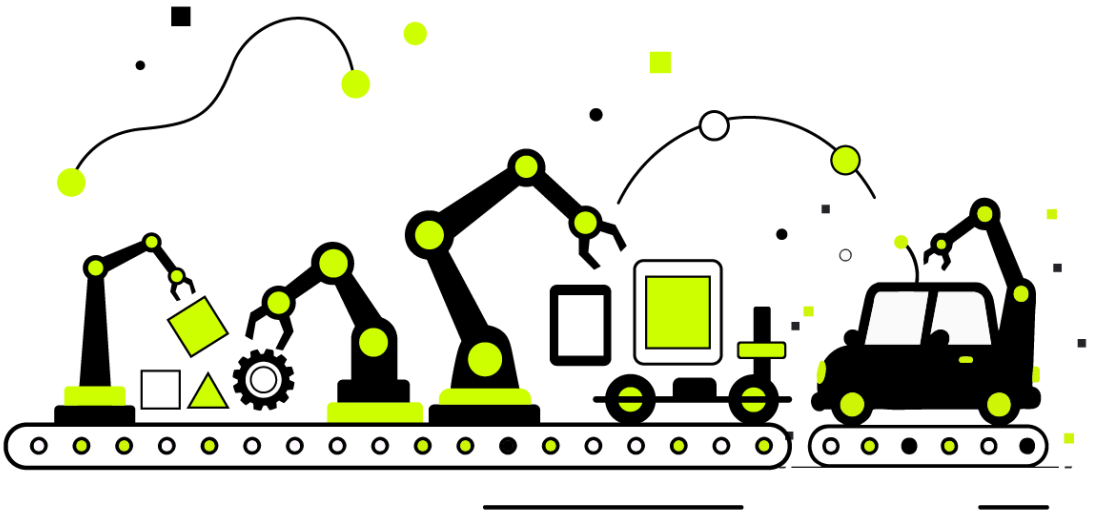
ნეირონული ქსელი არის კომპიუტერული მოდელი, რომელიც ადამიანის ტვინის სტრუქტურითაა შთაგონებული. ჩვენს ტვინში მილიარდობით ნეირონია, რომლებიც ერთმანეთთანაა დაკავშირებული და ინფორმაციას ამუშავებენ. ნეირონული ქსელიც მსგავსი პრინციპით მუშაობს. ის შედგება ხელოვნური „ნეირონებისგან“, რომლებიც ერთმანეთს უკავშირდება.



1.1. ადამიანის ტვინის სტრუქტურით შთაგონებული მოდელი

ნეირონული ქსელის იდეა ადამიანის ტვინის მუშაობის პრინციპიდან მომდინარეობს. ჩვენი ტვინი მილიარდობით ნეირონისგან შედგება, რომლებიც ერთმანეთთან დაკავშირებული არიან და ერთმანეთში სიგნალებს ცვლიან. სწორედ ამ მრავალსაფეხურიანი დამუშავების შედეგად ხდება ინფორმაციის ანალიზი. ხელოვნური ნეირონული ქსელის შემთხვევაში, „ხელოვნური ნეირონები“ (ანუ კოდის პატარა ნაწილები) ასევე დაკავშირებულია ერთმანეთთან. თითოეული ნეირონი იღებს სიგნალს, ამუშავებს მას და გადასცემს მომდევნო ნეირონს.

ანალოგია: წარმოიდგინე, რომ თამაშობ „ტელეფონს“ დიდ ჯგუფთან ერთად. პირველი ადამიანი (შესასვლელი ნეირონი) იღებს ინფორმაციას (მაგალითად, წინადადებას) და გადასცემს მას მეორეს, შემდეგ კი – მესამეს. ყოველი ადამიანი ამუშავებს მიღებულ ინფორმაციას და გადასცემს მას შემდეგს, სანამ ბოლო ადამიანი (გამომავალი ნეირონი) არ გამოიტანს საბოლოო დასკვნას. ზუსტად ამ პრინციპით მუშაობს ნეირონული ქსელი: ინფორმაცია გადის სხვადასხვა „ფენაში“ , სადაც ის ეტაპობრივად მუშავდება და ანალიზდება მანამ, სანამ საბოლოო პასუხი არ ჩამოყალიბდება.



1.2. ანალოგია: კომპანია, სადაც სხვადასხვა დეპარტამენტი ერთად მუშაობს რთული პრობლემის გადასაჭრელად

წარმოიდგინე, რომ ნეირონული ქსელი არის დიდი კომპანიასხვადასხვა დეპარტამენტით (**ფენა**) დათანამშრომლით (**ნეირონი**).

- **შესასვლელი ფენა (Input Layer):** ეს არის პირველი დეპარტამენტი, რომელიც იღებს მომხმარებლისგან შემოსულ ინფორმაციას (მაგ., სიტყვა „გამარჯობა“). ეს ფენა პასუხისმგებელია მონაცემების მიღებაზე. ამ მონაცემების დამუშავებაზე და შემდეგი ეტაპისთვის გადაცემაზე.
- **დამალული ფენები (Hidden Layers):** ეს არის შუალედური დეპარტამენტები, სადაც ინფორმაციის დამუშავება გრძელდება. თითოეული თანამშრომელი (ნეირონი) ინფორმაციას წინა ფენიდან იღებს, კვლავ ამუშავებს მას და გადასცემს მომდევნო ფენას. რაც უფრო მეტი ასეთი ფენა არსებობს, მით უფრო კომპლექსური ინფორმაციის დამუშავება შეუძლია ქსელს.
- **გამომავალი ფენა (Output Layer):** ეს არის ბოლო დეპარტამენტი, რომელიც საბოლოო გადაწყვეტილებას იღებს (მაგ., პასუხი „მისალმება“). ამ ფენის ამოცანაა, წარმოადგინოს საბოლოო შედეგი.

ნეირონები ერთმანეთთან არიან დაკავშირებულები და თითოეულ კავშირს გარკვეული კოეფიციენტი (**Weights**) აქვს მინიჭებული. სწავლის პროცესში კოეფიციენტის რიცხვითი

მნიშვნელობა იცვლება. სწორედ, ეს კოეფიციენტები განსაზღვრავენ იმას, თუ რა ინფორმაციას გადასცემს ერთი ნეირონი მეორეს.

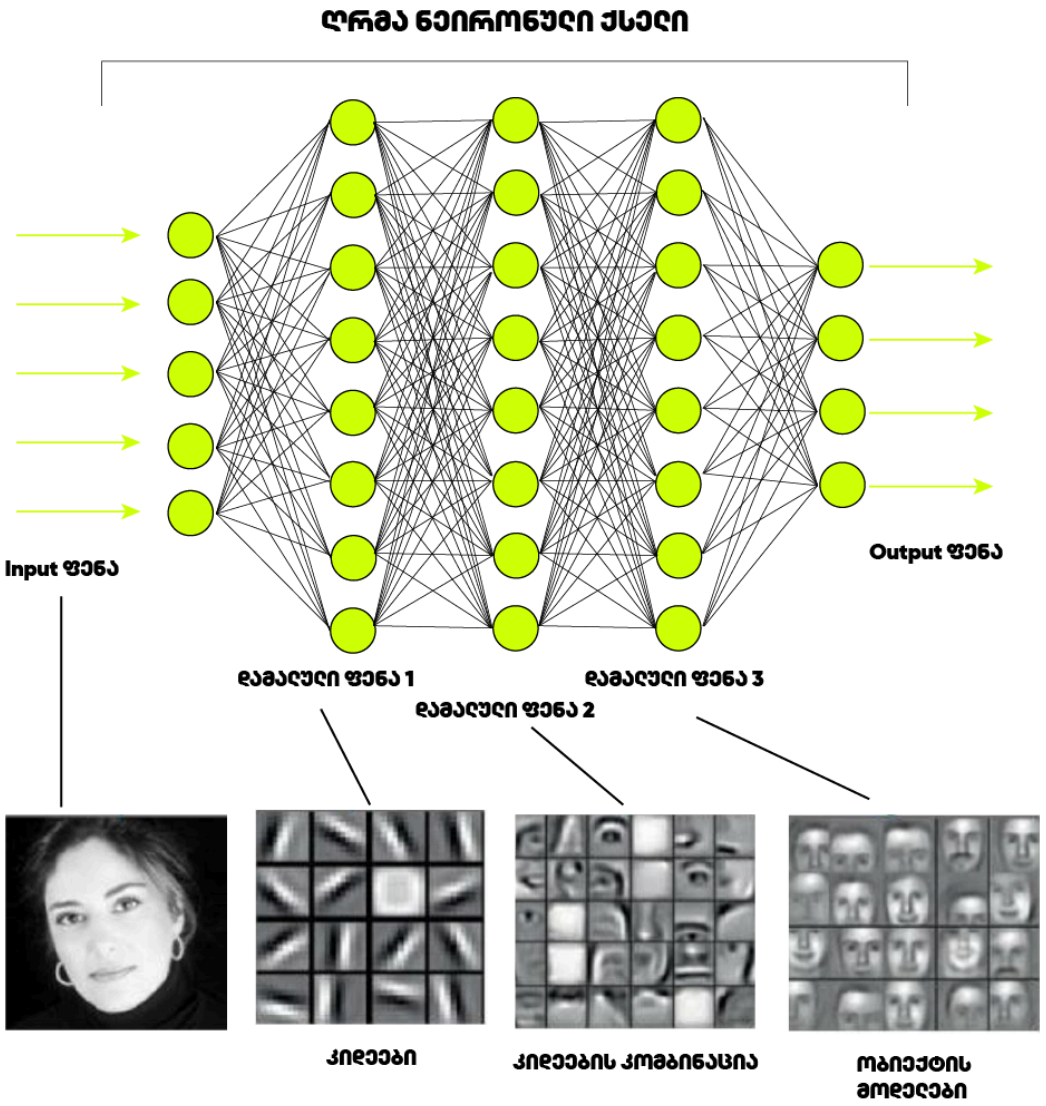
2. რა არის ღრმა სწავლება (Deep Learning)?

ღრმა სწავლება არის ხელოვნური ინტელექტის ქვესფერო, რომელიც ნეირონულ ქსელებზეა დაფუძნებული. თუ ჩვეულებრივ ნეირონულ ქსელს მხოლოდ რამდენიმე ფენა აქვს, ღრმა სწავლება იყენებს მრავალ, ასობით ან ათასობით ფენას, რათა შეძლოს მონაცემებში ძალიან რთული და აბსტრაქტული კანონზომიერებების აღმოჩენა. ღრმა სწავლების წყალობითაა, რომ დღევანდელ ხელოვნურ ინტელექტს შეუძლია სახის ამოცნობა, ადამიანურ ენაზე საუბარი და რთული პრობლემების გადაჭრა.

2.1. მრავალი ფენის მქონე ღრმა ნეირონული ქსელები

წარმოიდგინე, რომ ხარ დეტექტივი და ადამიანის ამოცნობას მხოლოდ მისი სურათის საფუძველზე ცდილობ.

- **პირველ ეტაპზე** (პირველი ფენა) შენ ამჩნევ მხოლოდ უმარტივეს დეტალებს: ხაზებს, კუთხეებსა და ფერებს.
- **მეორე ეტაპზე** (მეორე ფენა) ამ ხაზებიდან და კუთხეებიდან შენ უკვე აწყობ უფრო რთულ ფორმებს: თვალებს, ცხვირსა და პირს.
- **მესამე ეტაპზე** (მესამე ფენა) ამ ფორმებიდან შენ უკვე ქმნი სახის გამოსახულებას და იწყებ პიროვნების ამოცნობას.



ღრმა ნეირონული ქსელიც ზუსტად ასე მუშაობს: ყოველი ახალი ფენა ამუშავებს წინა ფენიდან მიღებულ ინფორმაციას და მასში უფრო რთულ კანონზომიერებებს პოულობს. სწორედ ამიტომ, რაც უფრო მეტი ფენა აქვს ქსელს, მით უფრო ღრმა და დეტალური ანალიზის უნარი აქვს.

2.2. რატომ არის ღრმა სწავლება განსაკუთრებით ძლიერი ენასთან და სურათებთან მუშაობისას

წარმოიდგინე, რომ კომპანია, რომელზეც ადრე ვისაუბრეთ, ერთი დეპარტამენტის ნაცვლად რამდენიმე დეპარტამენტისგან შედგება. ასეთი მრავალფენიანი სტრუქტურა ნეირონულ ქსელს საშუალებას აძლევს, რომ რთული, აბსტრაქტული კანონზომიერებები გაიაზროს.

- **სურათის ანალიზისას:** ერთმა ფენამ შეიძლება ამოიცნოს წრეები და ფორმები, შემდეგმა - თვალები და ცხვირი, ხოლო ბოლომ - მთლიანი სახის გამოსახულება.
- **ენასთან მუშაობისას:** ასეთი ღრმა ქსელი კონტექსტს, გრამატიკასა და სინონიმებს უკეთესად აანალიზებს.

3. დიდი ენობრივი მოდელები (LLMs)

ChatGPT, Gemini და სხვა მსგავსი ხელოვნური ინტელექტის სისტემები სწორედ ღრმა სწავლების ერთ-ერთი უახლესი მიღწევის - **Transformer** არქიტექტურის საფუძველზე მუშაობენ. Transformer-ის არქიტექტურა საშუალებას აძლევს მოდელს, რომ არა მხოლოდ წინა სიტყვები, არამედ მთელი წინადადების კონტექსტი გაიგოს ერთდროულად.

3.1. Fine-tuning-ის კონცეფცია: როგორ ვასწავლოთ გენიოსს ახალი საგანი

წარმოიდგინე, რომ დიდი ენობრივი მოდელი არის გენიალური, მაგრამ ზოგადი განათლების მქონე ადამიანი. მას შეუძლია ნებისმიერ თემაზე საუბარი, მაგრამ არ არის არც ერთი სფეროს სპეციალისტი. **Fine-tuning-ი** არის პროცესი, როდესაც ასეთ მოდელს ვაწვდით კონკრეტულ, მცირე მონაცემთა ბაზას (მაგ., ჩვენი ანITა-ს პროექტის შესახებ) და ვავარჯიშებთ, რათა ის კონკრეტულ სფეროში სპეციალისტად იქცეს. ეს საშუალებას გვაძლევს, რომ ჩვენი მოდელი უფრო ეფექტური გახდეს კონკრეტული ამოცანების შესასრულებლად.

დავალება 12: ნეირონული ქსელის პრაქტიკული ვარჯიში.

ამ დავალებათა ნაკრების (15.1, 15.2, 15.3) მიზანია, მოგცეთ სრული პრაქტიკული გამოცდილება **TensorFlow-ს საბაზისო პროგრამის შექმნაში.**

თქვენ სწავლობთ, როგორ შექმნათ უმარტივესი ნეირონული ქსელი, რომელიც რიცხვებს შორის დამალულ მათემატიკურ ფორმულას (`Y = 2X - 1`) თავისით პოულობს.

დავალება 12.1: შეცდომის პოვნა და გასწორება

ქვემოთ მოცემულ კოდში დაშვებულია ლოგიკური შეცდომა მონაცემების შექმნისას. `X` და `Y` მასივებში ელემენტების რაოდენობა არ ემთხვევა ერთმანეთს, რაც `model.fit()` ბრძანების გამოძახებისას გამოიწვევს შეცდომას. იპოვე, სად არის შეუსაბამობა და გაასწორე ის.

```
# შეცდომით დაწერილი კოდი
import tensorflow as tf
import numpy as np

# ნაბიჯი 2: სამუშაო მონაცემების შექმნა
# ყურადღება მიაქციე ელემენტების რაოდენობას თითოეულ მასივში!
X = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float) # აქ 6 ელემენტია
```

```
Y = np.array([-3.0, -1.0, 1.0, 3.0, 5.0], dtype=float)      # აქ კი 5 ელემენტია

# კოდის დანარჩენი ნაწილი სინტაქსურად სწორია, მაგრამ მონაცემების შეცდომის გამო ვერ იმუშავებს
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, input_shape=[1])
])

model.compile(optimizer='sgd', loss='mean_squared_error')

# პროგრამა შეცდომას დაუშვებს ამ ხაზზე, რადგან X და Y მასივების ზომა არ ემთხვევა
model.fit(X, Y, epochs=500, verbose=0)

input_data = np.array([10.0])
prediction = model.predict(input_data)

print(f"მოდელის პროგნოზი: {prediction[0][0]}")
```

დავალება 12.2: კოდის დასრულება

მოცემულ კოდს აკლია ერთ-ერთი უმნიშვნელოვანესი ნაბიჯი — **მოდელის კომპილაცია (`model.compile`)**. ეს ნაბიჯი ამზადებს მოდელს გაწვრთნისთვის. დაამატე გამოტოვებული კოდის ხაზი, სადაც ოპტიმიზატორად გამოიყენებ `'sgd'`-ს, ხოლო დანაკარგის ფუნქციად — `'mean_squared_error'`-ს.

```
# კოდი, რომელიც უნდა დაასრულო
import tensorflow as tf
import numpy as np

X = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
Y = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)

model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, input_shape=[1])
])
```

```
# დაწერე შენი კოდი აქ: მოამზადე მოდელი წვრთნისთვის .compile() მეთოდის გამოყენებით
# მიუთითე optimizer და loss პარამეტრები.


# კოდის დანარჩენი ნაწილი დასრულებულია
model.fit(X, Y, epochs=500, verbose=0)

input_data = np.array([10.0])
prediction = model.predict(input_data)

print(f"მოდელის პროგნოზი: {prediction[0][0]}")
```

დავალება 12.3: შეიმუშავე პროგრამული კოდი

დაწერე პროგრამული კოდი, შექმენი პროგრამა, რომელიც:

1. გააკეთებს `tensorflow`-სა და `numpy`-ს იმპორტს.
2. შექმნის `X` და `Y` `numpy` მასივებს (იგივე რიცხვებით, რაც მაგალითშია).
3. ააწყობს `Sequential` მოდელს ერთი `Dense` ფენით, რომელსაც 1 ნეირონი და `input_shape=[1]` ექნება.
4. გააკეთებს მოდელის კომპილაციას `sgd` ოპტიმიზატორითა და `mean_squared_error` დანაკარგის ფუნქციით.
5. გაწვრთნის მოდელს 500 ეპოქის განმავლობაში.
6. გააკეთებს პროგნოზს `X=10.0`-სთვის და დაბეჭდავს შედეგს.

```
# დაწერე შენი კოდი აქ:
```

სწორი პასუხი (პროგრამული კოდი სრულად):

```
# ნაბიჯი 1: საჭირო ბიბლიოთეკების იმპორტი
import tensorflow as tf
import numpy as np

# ნაბიჯი 2: სამუშაო მონაცემების შექმნა
# ეს არის X და Y წყვილები, რომლებითაც მოდელი სწავლობს ფორმულას.
X = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=float)
Y = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=float)
```

```
# ნაბიჯი 3: მოდელის შექმნა
# ვქმნით უმარტივეს ნეირონულ ქსელს: ერთი ფენა ერთი ნეირონით.
# input_shape=[1] ნიშნავს, რომ თითოეული შემავალი მონაცემი (X) ერთი რიცხვია.
model = tf.keras.Sequential([
    tf.keras.layers.Dense(units=1, input_shape=[1])
])

# ნაბიჯი 4: მოდელის კომპილაცია
# ვამზადებთ მოდელს სწავლისთვის: ვუთითებთ სწავლის სტრატეგიას (optimizer)
# და შეცდომის საზომს (loss).
model.compile(optimizer='sgd', loss='mean_squared_error')

# ნაბიჯი 5: მოდელის გაწვრთნა
# fit ბრძანებით მოდელი 500-ჯერ (epochs=500) გადახედავს მონაცემებს
# და შეეცდება, მინიმუმამდე დაიყვანოს შეცდომა.
model.fit(X, Y, epochs=500, verbose=0) # verbose=0 მაღავს გაწვრთნის დეტალურ პროცესს

# ნაბიჯი 6: პროგნოზის გაკეთება
# ვიყენებთ გაწვრთნილ მოდელს, რომ ვიპოვოთ Y ახალი X=10.0-სთვის.
# მნიშვნელოვანია, რომ predict მეთოდს მონაცემი გადაეცეს სიის ან მასივის სახით.
new_X = 10.0
# ეს აძლევს მონაცემს იმ ფორმას, რასაც TensorFlow ელოდება
input_data = np.array([new_X])

prediction = model.predict(input_data)

print("---------------------------------")
print(f"ვაკეთებთ პროგნოზს X = {new_X}-სთვის...")
print(f"მოდელის პროგნოზი: {prediction[0][0]}")
print("---------------------------------")
```