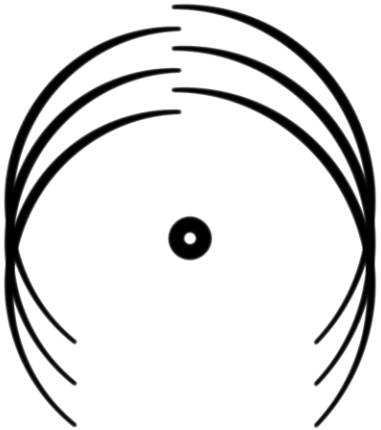
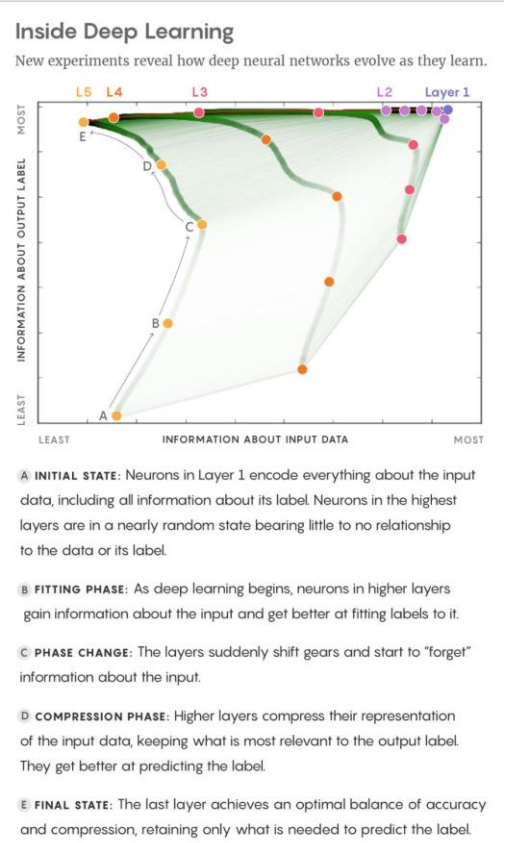
Derin Öğrenme ile Renormalizasyon Grupları Arası Haritalama Atölyesi

Göktuğ İslamoğlu
İTÜ İnşaat Müh.



Atölye İçin Gerekenler

- Temel Python bilgisi
- Anaconda Navigator (Python 2.7 Sürümü)
- PyCX Paketi http://pycx.sourceforge.net/
- https://github.com/goktu/AR2D2

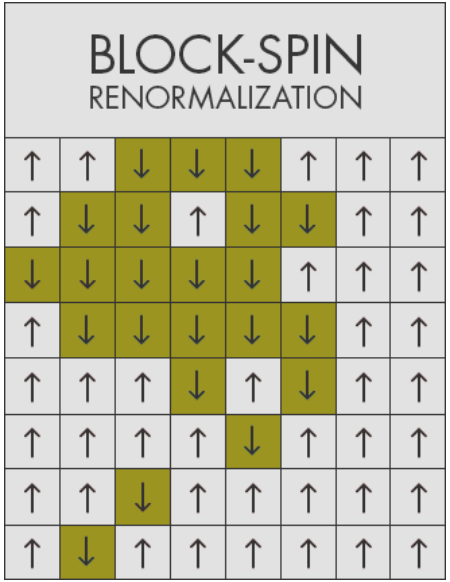


Derin Öğrenmenin Kara Kutusu

Derin öğrenmenin başarısı yaratıcılarını bile şaşırtırken, bu sistemlerin nasıl çalıştığını açıklayan temel bir prensip kesin olarak bilinmemektedir. Derin öğrenmenin, beyin davranışlarının taklit edildiği düşünülmektedir.

Tishby, "bilgi darboğazı" teorisini öne sürmüştür. Derin öğrenme sinir ağları eğitilirken, algoritmanın istenilen bilgiye yoğunlaştığı ve istenilmeyen bilgiyi unuttuğu, Hinton gibi uzmanlar tarafından da tutarlı bulunmaktadır.

https://www.quantamagazine.org/new-theory-cracks-open-the-black-box-of-deep-learning-20170921/



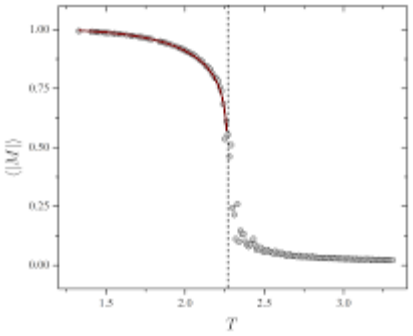
Renormalizasyon Grubu (RG)

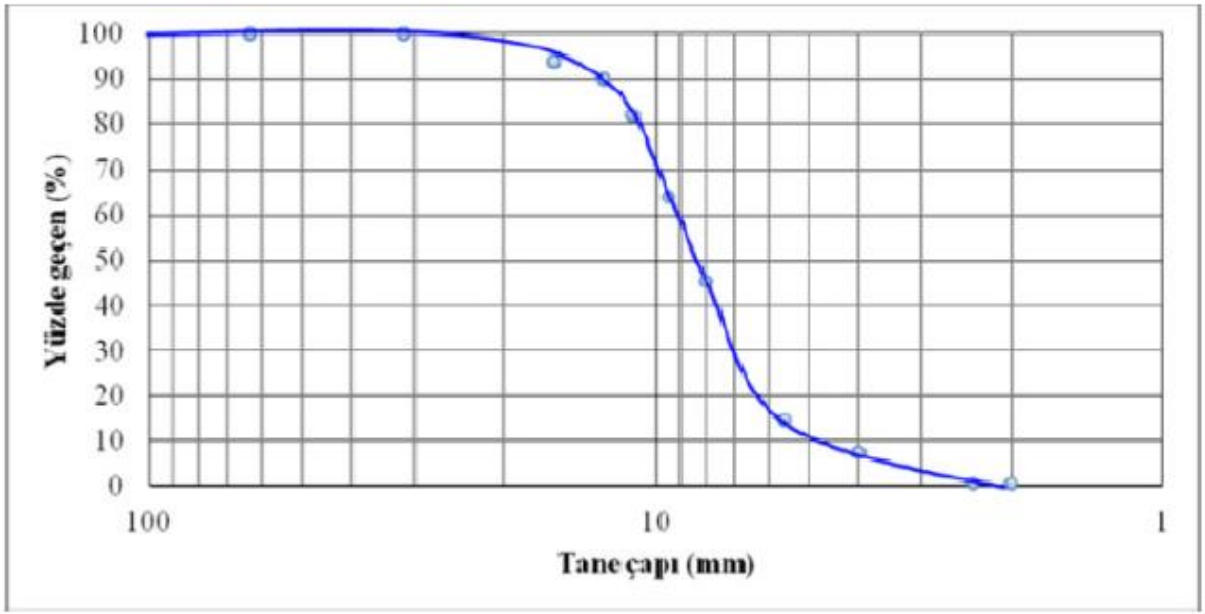
Scwab ve Mehta, fizikte yaygınlıkla kullanılan bir yöntem olan “renormalizasyon grupları”nın, derin öğrenmedeki darboğazda bahsedilen, bilgiyi hatırlama ve unutma davranışlarını açıkladığını yayınladığı makalede gösterdi.

“Coarse-graining”, yani “iri daneleme” yaklaşımı ile, sistemin en detaylı ifade edildiği “mikroskopik” halinden, en özet ifade edildiği “makroskopik” haline, yeterince iyi bir sonuç alarak ulaşabiliriz.

https://www.quantamagazine.org/deep-learning-relies-on-renormalization-physicists-find-20141204/

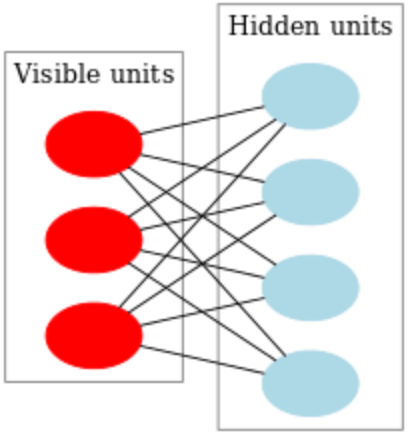
İnşaatçı Gözüyle İri Daneleme (Coarse-Grain)

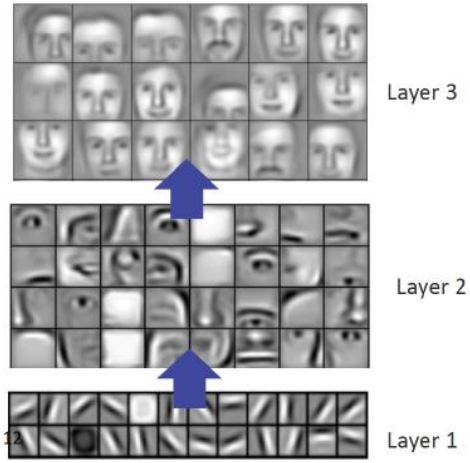




Şekil 3. Geri dolgu malzemesi ortalama granülometre eğrisi

Yüksel Z.T., Çevik E.Ö. "Investigation of Seismic Behavior of Vertical Block Type Quay Wall.", YTÜ PhD Thesis, 2016.





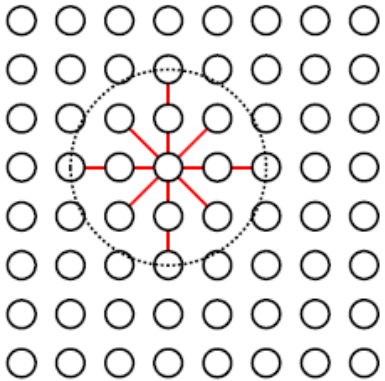
Kısıtlı Boltzmann Makinesi (RBM)

Geoffrey Hinton'ın Kısıtlı Boltzmann makineleri için hızlı öğrenme algoritmalarını geliştirmesi, 2000'lerde derin öğrenme araştırmalarında patlama yaşattı.

Kısıtlı Boltzmann makineleri, görünür ve gizli katmanlar arasında bağlantı kurulmasını sağlarken, kısıtlı olmayan Boltzmann makinelerinin aksine gizli katmanlar arası iletişim engellenir. Bu da gradyan-temelli eğitim algoritmalarının verimini arttırır.

https://en.wikipeda.org/wiki/Restricted_Boltzmann_machine

Atölyenin Amacı

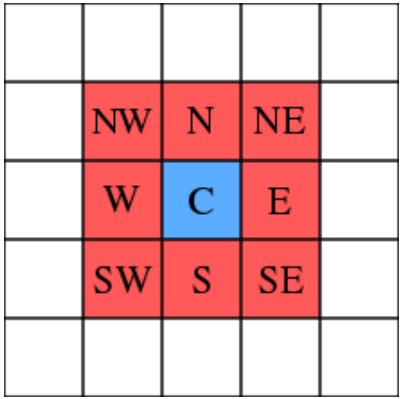


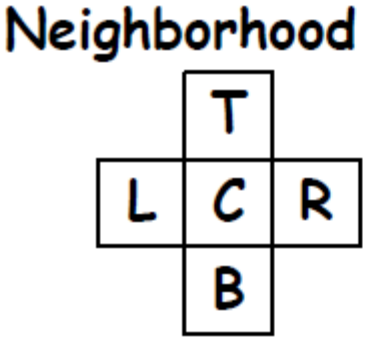
Kısıtlı Boltzmann Makineleri kullanılarak gerçekleştirilen renormalizasyon iri daneleme işlemi ile edinilen bilgi darboğazını, Hücresel Otomaton (Cellular Automaton) kullanarak incelemektir.

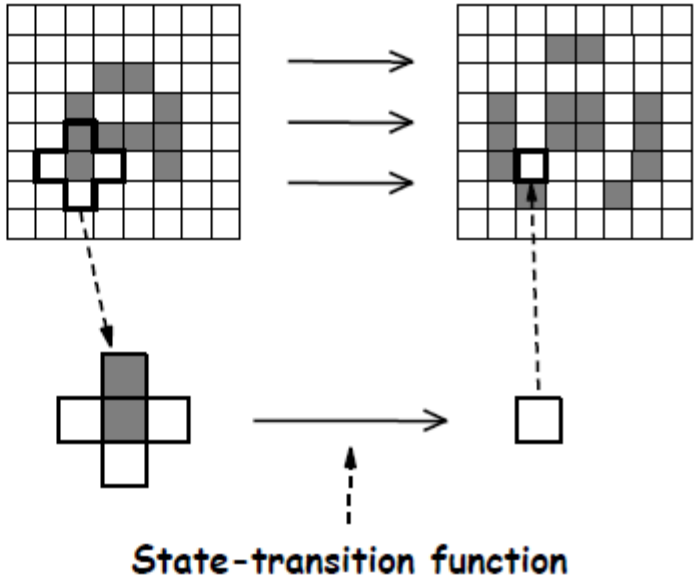
Hücresel Otomaton: Sonlu ve sınırlı bir ızgarada, eş zamanlı güncellenen bir durum geçiş fonksiyonunun, bir hücrenin komşularına göre hücre durumunu belirlemesi.

Sayama, Hiroki. “Introduction to the Modeling and Analysis of Complex Systems, Binghamton University, Open SUNY Textbooks

Hücresel Otomata Nasıl Çalışır?







Sayama, Hiroki. “Introduction to the Modeling and Analysis of Complex Systems, Binghamton University, Open SUNY Textbooks

1. Atölye: Game of Life

Hücresel Otomata'ya giriş için en meşhur HO modellerinden biri olan Game of Life'ı inceleyeceğiz.

Ünlü İngiliz matematikçi John Horton Conway tarafından geliştirilen GoL, aşırı basit bir kod ile son derece karmaşık davranışlar ortaya çıkarmasıyla meşhur oldu.

Anaconda Navigator'ı açıp, Spyder IDE'yi başlatalım.

Anaconda'nın Python sürümünün 2.7 olması önemli!

1. Atölye: Game of Life

pycx-0.32/realtime-simulation-template.py

```
import matplotlib
matplotlib.use('TkAgg')

##=====================================
## Section 1: Import Modules
##=====================================

# i.e., import pylab as PL

##=====================================
## Section 2: Define Model Parameters
##=====================================

# i.e., n = 10000, RD.seed(), etc.

##=====================================
## Section 3: Define Three Functions
##=====================================
```

Game of Life: Yapılandırma

Pylab modülünü çağırıyoruz.

Kare ızgaranın boyutunu ve dağılım olasılığını giriyoruz.

```
import matplotlib
matplotlib.use('TkAgg')

from pylab import *

L = 100
p = .3
```

Game of Life: Başlatma

c ve n global parametrelerini tanımlıyoruz.

c: ızgara için ilk dizi; nc: ızgara için sonraki dizi

c için rastgele dağılım belirleyip, 0 ya da 1 veriyoruz.

```
def init():
    global c, nc
    c = zeros([L,L])
    for x in xrange(L):
        for y in xrange(L):
            c[x,y] = 1 if random() < p else 0
    nc = zeros([L,L])
```

Game of Life: Çizim

Eksenleri cla() ile temizleyip, imshow(c) ile dizileri çiziyoruz.

```
def draw():
    cla()
    imshow(c)
```

Game of Life: Komşu Hücreler

c dizisindeki x hücresinin Moore komşu dizisini tarıyoruz.

Tüm hücrelerin değerlerinin toplamı, kaç tane 1, yani “yaşayan” komşu hücre olduğunu veriyor.

En sonunda iki kere toplamamak adına c[x,y]’i çıkartıyoruz.

```
def number_of_living_neighbors(x,y):
    count = 0
    for dx in range(-1, 2):
        for dy in range(-1, 2):
            count += c[(x+dx)%L, (y+dy)%L]
            print count
    return count - c[x,y]
```

Game of Life: Adımlar

EĞER hücrenin değeri 0 ise: 3 tane yaşayan komşu varsa hücre 1 olsun, yoksa 0 kalsın

YOKSA 2'den az komşu varsa ya da 3'ten fazla komşu varsa hücre 0 olsun, YOKSA hücre 1 olsun.

```
def step():
    global c, nc
    for x in xrange(L):
        for y in xrange(L):
            n = number_of_living_neighbors(x,y)
            if c[x,y] == 0:
                nc[x,y] = 1 if n == 3 else 0
            else:
                nc[x,y] = 0 if n < 2 or n > 3 else 1
    c, nc = nc, c
```

Game of Life: Adımlar (2)

Bu koşulların Hayat Oyunu içindeki anlamı:

3 tane yaşayan komşu varsa nüfus artar;

2'den az komşu varsa açlıktan ölür;

3'ten fazla komşu varsa rekabetten ölür;

YOKSA nüfus artar.

En sondaki *c, nc = nc, c* ifadesi, c'deki değişikliklerin adım bittikten sonraki dizi olan nc'ye yazılmasını sağlar.

Game of Life: Çalıştırma

setSomeParameter fonksiyonunu sildikten sonra, Anaconda Navigator'da Spyder'da çalıştırıyoruz.

Çalıştırırken, yazdığımız kodun konumunun, "pycxsimulator.py" dosyası ile aynı olması gerekiyor.

Spyder ayarlarından pencerede çalıştırmayı seçiyoruz.

Devamlı olarak çalıştırıp, sistemin davranışını inceleyelim. Sistemin dağılım olasılığı p'yi değiştirip, sistemin yeni davranışını gözlemleyelim.

Ising Modeli Hücresel Otomatonu

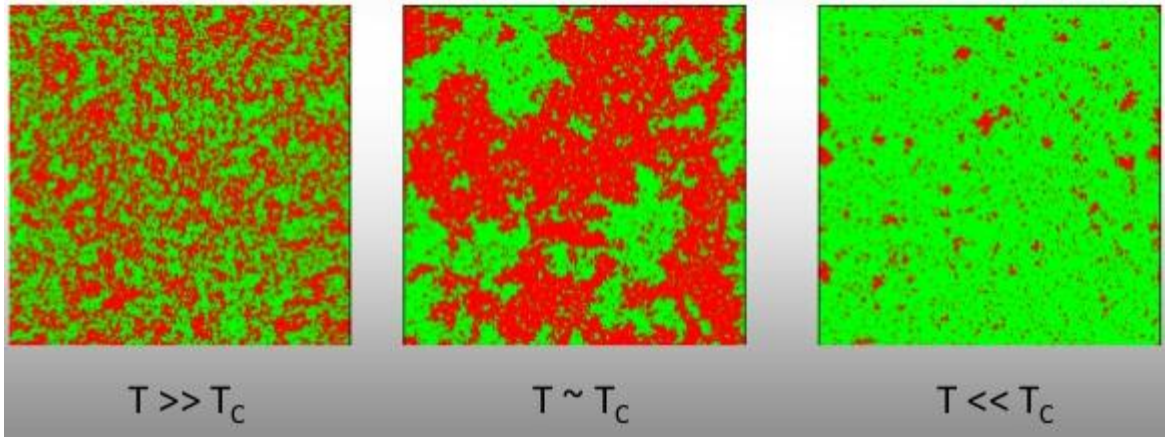
Hücresel otomatonlara ısındıktan sonra, renormalizasyon gruplarının nasıl HO'larca ifade edilebileceğine bakalım.

Ising modeli, Ernst Ising'in 1924 yılında geliştirdiği, atom dönülerinin (spin) manyetik dipol momentlerini ifade eden sonlu bir sistemdir.

Ising'in kendi modeli, 1 boyulu dizilimlerde faz değişimi olmayacağını gösterirken, 2 boyutlu ızgaralarda faz değişiminin olduğu, Peierls tarafından gösterildi ve kritik değer Onsager tarafından 1945 yılında bulundu.

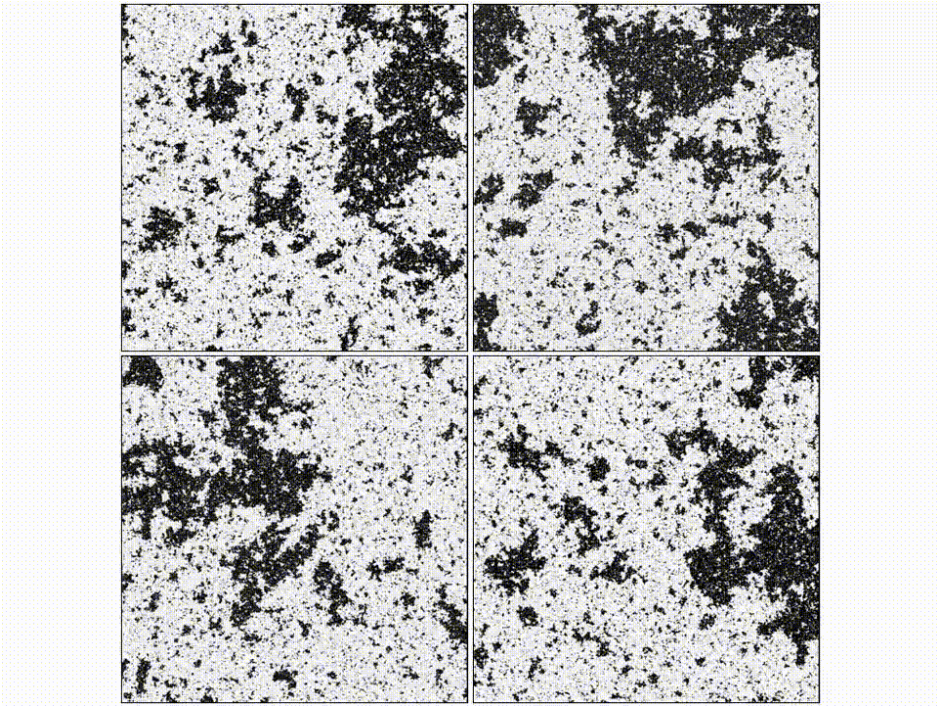
Ising Modelinde Ölçeklenebilirlik

2 boyutlu modelde, sistemin ısısı kritik eşik olan Tc'ye ulaştığında, sistem bütün ölçeklerde aynı davranışı göstermeye başlar, yani faz değişimi görülür.



https://www.slideshare.net/BenjaminBlock1/isinggpu

Ising Modelinde Fraktal Davranış

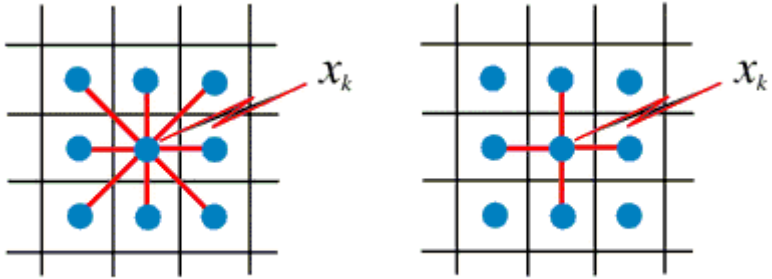


https://gfycat.com/gifs/detail/dizzyagedhomalocephale

HO ile Ising Modeli'nin RG'si

Ising modelinin HO'sunu elde etmeye çalışırken, RG yöntemini kullanabilmek için faz değişimi oluşturmamız gerekiyor.

Ising modeli ile HO arasındaki bağıntı: komşu ilişkileri



https://link.springer.com/chapter/10.1007/978-3-319-24871-4_25

2. Atölye: Ising Modeli

1. atölyede yazdığımız kod dosyasını kopyalayalım.

Bu sefer Numpy modülünü de çağırıyoruz.

```
import matplotlib
matplotlib.use('TkAgg')

from pylab import *
import numpy as np

L = 100  # size of 2D lattice: LxL

p = 0.5
```

Ising Modeli: Üst ve Alt Komşular

init ve *draw* fonksiyonları aynı kalıyor. draw'un altında, üst ve alt komşuların canlı hücre sayısını hesaplayalım.

```
def number_of_upper_neighbors(x, y):
    upper_count = 0
    for dx in range(-1, 2):
        upper_count += c[(x + dx) % L, (y + 1) % L]
        # print upper_count
    return upper_count

def number_of_lower_neighbors(x, y):
    lower_count = 0
    for dx in range(-1, 2):
        lower_count += c[(x + dx) % L, (y - 1) % L]
        # print lower_count
    return lower_count
```

Ising Modeli: Sağ ve Sol Komşular

Sağ ve Sol komşuların canlı hücre sayısını hesaplayalım.

```
def number_of_right_neighbors(x, y):
    right_count = 0
    for dy in range(-1, 2):
        right_count += c[(x + 1) % L, (y + dy) % L]
        # print right_count
    return right_count

def number_of_left_neighbors(x, y):
    left_count = 0
    for dy in range(-1, 2):
        left_count += c[(x - 1) % L, (y + dy) % L]
        # print left_count
    return left_count
```

Ising Modeli: Neumann Komşuları

Neumann komşularının canlı hücre sayısını hesaplayalım.

```
def number_of_Neumann_neighbors(x, y):
    Vertical_count = 0
    Horizontal_count = 0
    for dy in range(-1, 2):
        Vertical_count += c[x, (y + dy) % L]
        # print Vertical_count
    for dx in range(-1, 2):
        Horizontal_count += c[(x + dx) % L, y]
        # print Horizontal_count
    return Vertical_count + Horizontal_count - c[x, y]
```

Ising Modeli: Moore Komşuları

Moore komşularının canlı hücre sayısını hesaplayalım.

```
def number_of_Moore_neighbors(x, y):
    Moore_count = 0
    for dx in range(-1, 2):
        for dy in range(-1, 2):
            Moore_count += c[(x + dx) % L, (y + dy) % L]
            # print Moore_count
    return Moore_count - c[x, y]
```

Bütün komşu tarama fonksiyonları tamamlandığına göre, adımlara geçebiliriz!

Ising Modeli: Adımlar (1)

```
def step():
    global c, nc, array, array0, array1, count0, count1
    count0 = 0
    count1 = 0
    i = 0
    j = 0
    array = []
    array0 = []
    array1 = []
    for x in xrange(L):
        for y in xrange(L):
            array.append(c[x, y])
            g = number_of_Moore_neighbors(x, y)
            if c[x, y] == 0:
                nc[x, y] = 0 if g <= 6 else 1  # nearest-neightbor parameter
                array0.append(c[x, y])
            elif c[x, y] == 1:
                array1.append(c[x, y])
```

Ising Modeli: Adımlar (2)

```
elif c[x, y] == 1:
    array1.append(c[x, y])
    for z in range(-1, 2):
        # bottleneck configuration due to arrested state
        m = number_of_upper_neighbors(x, y)
        if m == 1:
            nc[x, (y + 1) % L] = 1

        n = number_of_lower_neighbors(x, y)
        if n == 1:
            nc[x, (y - 1) % L] = 1

        k = number_of_right_neighbors(x, y)
        if k == 0 and (m <= 1 or n <= 1):
            nc[(x + 1) % L, (y + z) % L] = 1

        l = number_of_left_neighbors(x, y)
        if l == 1 and (m > 1 or n > 1):
            nc[(x - 1) % L, (y + z) % L] = 0

        h = number_of_Neumann_neighbors(x, y)
        if h >= 1:
            # nearest-neighbor parameter
            nc[x, y] = 1 if g <= 6 else 0
```

Ising Modeli: Adımlar (2)

```
elif c[x, y] == 1:
    array1.append(c[x, y])
    for z in range(-1, 2):
        # bottleneck configuration due to arrested state
        m = number_of_upper_neighbors(x, y)
        if m == 1:
            nc[x, (y + 1) % L] = 1

        n = number_of_lower_neighbors(x, y)
        if n == 1:
            nc[x, (y - 1) % L] = 1

        k = number_of_right_neighbors(x, y)
        if k == 0 and (m <= 1 or n <= 1):
            nc[(x + 1) % L, (y + z) % L] = 1

        l = number_of_left_neighbors(x, y)
        if l == 1 and (m > 1 or n > 1):
            nc[(x - 1) % L, (y + z) % L] = 0

        h = number_of_Neumann_neighbors(x, y)
        if h >= 1:
            # nearest-neighbor parameter
            nc[x, y] = 1 if g <= 6 else 0
```

Ising Modeli: Adımlar (3)

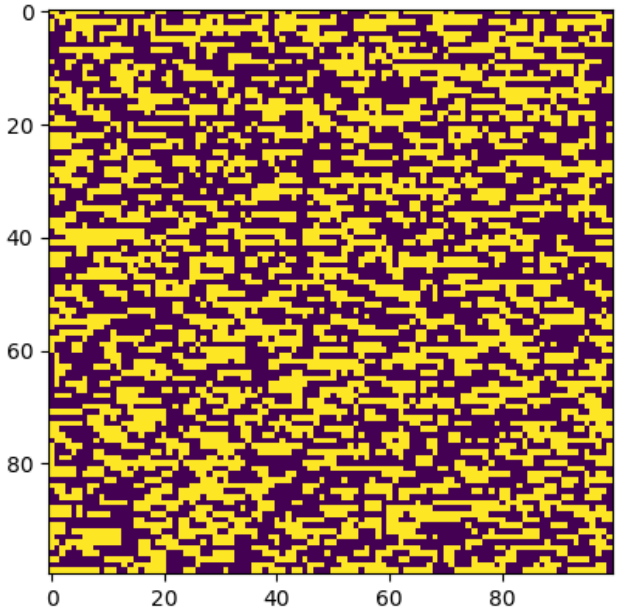
```
            i += 1
            j += g
    count0 = len(array0)  # total count of cells with value 0 in each step
    count1 = len(array1)  # total count of cells with value 1 in each step

    print count1

    c, nc = nc, c

import pycxsimulator
pycxsimulator.GUI(title='My Simulator', interval=0,
                  parameterSetters=[]).start(func=[init, draw, step])
```

Ising Modeli: Çalıştırma



Ising Modeli: Ödüllü Soru 1

Bu adaları, komşu sayısını y olarak alırsak, y = x(1-x) fonksiyonuna nasıl yerleştirerek uzatabiliriz?

İPUCU: Yukarıda Moore komşularını g'ye tanımladık

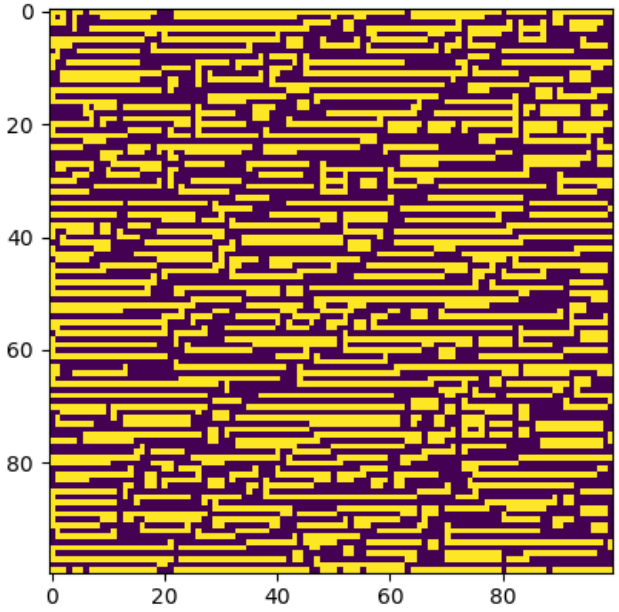
Ising Modeli: Ödüllü Soru 1

Bu adaları, komşu sayısını y olarak alırsak, y = x(1-x) fonksiyonuna nasıl yerleştirerek uzatabiliriz?

İPUCU: Yukarıda Moore komşularını g'ye tanımladık

```
if g / 8 > (1 - p) * p:  # next-nearest neighbor asymptote
    nc[(x + 1) % L, y] = 1
elif g / 8 < (1 - p) * p:
    nc[(x - 1) % L, y] = 1
else:
    nc[x, y] = 1
```

Ising Modeli: Çalıştırma (2)



Ising Modeli: Ödüllü Soru (2)

Sistemin üst ve alt olmak üzere iki tane sınır noktası bulunuyor.

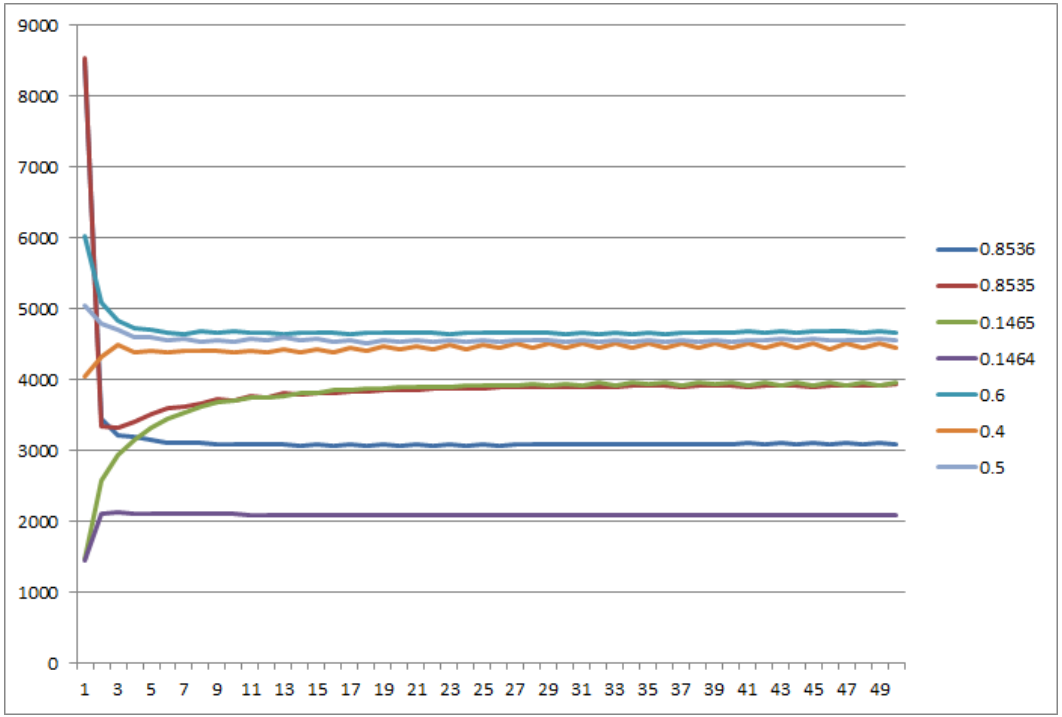
p olasılığını 0.8535 ve 0.8536 olarak değiştirip, 20 adımdan sonra çıktı alınan canlı hücre değerini karşılaştırın.

p olasılığını 0.1464 ve 0.1465 olarak değiştirip, 20 adımdan sonra çıktı alınan canlı hücre değerlerini karşılaştırın.

Bu sınır değerlerinin tam ifadesi nedir?

İPUCU: 1) 0.5 + … 2) 0.5 - …

Ising Modeli: Darboğaz



Katıldığınız için teşekkürler!

Göktuğ İslamoğlu
İTÜ İnşaat Müh.

goktug@captr.co