Çanakkale Onsekiz Mart Üniversitesi, Mühendislik Fakültesi,

Bilgisayar Mühendisliği Akademik Dönem 2022-2023

Ders: BLM-4014 Yapay Sinir Ağları/Bahar Dönemi

ARA SINAV SORU VE CEVAP KAĞIDI

Dersi Veren Öğretim Elemanı: Dr. Öğretim Üyesi Ulya Bayram

Öğrenci Ad: Esra ÇAVDAR

Öğrenci No: 190401083

Ödev Github Linki: https://github.com/Esra08/Yapay-Sinir-Aglari-Project

14 Nisan 2023

1 (Toplam 10 Puan) Hopfield Networks:

1.1 (2 Puan) ChatGPT'ye size Hopfield Network'ler ile ilgili detaylı bilgi vermesini isteyin. Sorduğunuz soruyu, ChatGPT'nin çıkarttığı cevabı aşağı koyun. Bu cevapta anlamadığınız terimleri de ChatGPT'ye sorup, sorduğunuz soruları ve ChatGPT'nin verdiği cevapları da sırayla aşağı koyun.

Hopfield ağları, yapay sinir ağlarının bir türüdür ve içerdiği öğrenme ve bellek işlevleri nedeniyle en popüler ağlardan biridir. İlk kez John Hopfield tarafından 1982'de tanıtılmıştır. Hopfield ağları, özellikle desen tanıma, desen tamamlama ve optimizasyon problemlerinde kullanılır.

Hopfield ağları, nöronları ve bu nöronların birbirleriyle bağlantılarını içerir. Bu ağlar, nöronlar arasındaki bağlantıların simetrik olduğu, yani ağın her iki yönünde de aynı bağlantıların olduğu bir özellik taşır. Bu özellik, ağın enerji fonksiyonunun yüksekliğinin, ağın tüm düğümlerindeki değerlerin yerini değiştirdiğinde aynı kalması anlamına gelir.

Hopfield ağları, unsupervised learning (denetimsiz öğrenme) yöntemi kullanır. Bu, ağın eğitim verisi üzerinde öğrenme işleminin gerçekleştirildiği ancak bu verilerin hangi sınıfa ait olduklarının belirtilmediği bir öğrenme yöntemidir. Ağ, eğitim verilerini hafızasında saklar ve daha sonra, yeni girdilerle karşılaştırarak en yakın benzerliği bulmaya çalışır.

Hopfield ağları, bir takım avantajlara sahiptir. Öncelikle, nöronların doğrusal olmayan özellikleri sayesinde, ağ çoklu desenleri depolayabilir ve geri çağırabilir. Ayrıca, bir desenin bozulması durumunda, ağ, orijinal deseni geri çağırmak için otomatik olarak düzeltme işlemleri yapabilir. Bunun yanı sıra, Hopfield ağları, öğrenme işlemini gerçekleştirmek için bir öğrenme algoritması gerektirmediği için, oldukça basit bir şekilde uygulanabilir.

Hopfield ağlarının bir diğer önemli uygulaması da optimizasyon problemlerinin çözümüdür. Bu tür problemler, bir fonksiyonun minimum veya maksimum noktalarını bulmak gibi, birçok alanda karşımıza çıkar. Hopfield ağları, bu problemlerin çözümünde kullanılan simüle edilen tavlama (simulated annealing) gibi optimizasyon tekniklerine alternatif bir yöntem sunar.

Anlamadığım kavram:

Unsupervised learning (denetimsiz öğrenme): Denetimsiz öğrenme (unsupervised learning), makine öğrenmesindeki bir öğrenme şeklidir. Bu yöntemde, verilerin etiketlenmemesi yani sınıflandırılmamış olması durumunda kullanılır. Denetimsiz öğrenme, verilerdeki kalıpları keşfetmek için kullanılır ve verilerin daha iyi anlaşılması, yapılandırılması veya özetlenmesi için kullanılabilir.

Denetimsiz öğrenme, genellikle iki farklı amaç için kullanılır:

Kümeleme (clustering): Verilerdeki benzer özelliklere sahip veri noktalarını gruplamak için kullanılır. Bu sayede verilerdeki yapılar keşfedilerek daha iyi anlaşılabilir.

Boyut indirgeme (dimensionality reduction): Verilerin boyutunu azaltarak verilerdeki karmaşıklığı azaltır. Bu sayede veriler daha iyi anlaşılabilir ve daha hızlı işlenir.

Denetimsiz öğrenme algoritmaları arasında en yaygın olanlar; K-Means kümeleme algoritması, PCA (principal component analysis - temel bileşen analizi), Autoencoder gibi algoritmalardır. Bu algoritmalar verilerdeki yapıları belirlemek için kullanılır ve birçok farklı alanda, özellikle de veri analizi, görüntü işleme, biyoinformatik ve doğal dil işleme alanlarında kullanılır.

Yukarıda yer alan ChatGPT' den alınmış bilgiler üstü kapalı gibi geldiği için Google' da da arama yaptım. İncelediğim kaynaklar:

https://en.wikipedia.org/wiki/Hopfield_network

https://bilgisayarkavramlari.com/2008/10/19/hopfield-aglarinin-sayisallasmasi/

https://medium.com/@batincangurbuz/hopfield-a%C4%9F-modeli-hopfield-network-hn-ccf1548ca432

1.2 (8 Puan) ChatGPT'nin açıklamalarından ne anladığınızı buraya kendi kelimelerinizle özetleyin. Örneğin "Hopfield Network nedir? Ne işe yarar? Neden bilmemiz gerekir bunu? Gerçek hayatta kullanılıyor mu?" gibi. Anlamadığınız kısımlar varsa ve ChatGPT'nin açıklamaları yeterli gelmezse internet dahil farklı kaynaklardan araştırıp, bilgilerin doğruluğunu sorgulamakta serbestsiniz. Konuyu doğru anlayıp anlamamanız puana etki edecektir. Kullandığınız kaynakları da belirtin!

Yapay sinir ağlarından biri olan ve adını zamanında tanıtan kişiden alan Hopfield Network tek bir katmana sahip olup giriş ve çıkış katmanının aynı olduğunu anladım. Hopfield Network' te karakterleri vs depolayıp daha sonra bunları çağırması olarak 2 aşaması var. Sayısal işaret işleme dersinde grupça yaptığımız harfleri tanıma projesini örnek olarak düşündüm. Orada her gelen karakter karşılaştırmaya tabi tutularak benzediği sayının grubuna ekleniyor. Bu karakterlerin çeşitli şekilleri depolanıyor ve öğrenme yapılıyor. Herhangi bir karakter çağrıldığında ise karşılaştırma sonucunda hangisi sayıya benziyorsa çıktı olarak o sayı yazdırılıyor. Bunun haricinde Hopfield Network yüz okumada , ekonomide veya tıpta kullanılabilir. Tıpta mr sonucuna göre görülenlerin kanser teşhisinde kullanılabilir. Başka bir yönden bakarsak Hopfield Network optimizasyon için de kullanılır. Maximumu arıyorsak en iyi maksimumu, minimumu arıyorsak en iyi minimumu bulmaya çalışır. Örneğin ekonomide özellikle optimizasyon problemlerinin çözümünde ve tahmin yapmak için kullanılır. Kötü bir örnek olacak belki ama portfköy optimizasyonu için kullanılıyor olabilir. Riski en aza indirmek için diğer portfköylerle karşılaştırma yaparak tahminlerde bulunuyordur. Müşterilerde bu tahminlere göre seçim yapıyor diyebiliriz.

Denetimsiz öğrenme kavramını görünce burada bilmediğimi zannetmiştim ama ilk derste değindiğiniz içeriğini okuduğumda aklıma geldi. Öğrenme tiplerinden olan denetimsiz öğrenmeye yukarıda Hopfield Network açıklamamda zaten değinmiş oldum. Gelen karakteri/objeyi karşılaştırarak benzer olanları kümeliyor. Bide boyut indirgeme amacıyla kullanılır. Bu veri kümesindeki değişken sayısını azaltarak, verinin anlaşılmasını kolaylaştırır.

2 (Toplam 10 Puan) Boltzman Machine:

2.1 (2 Puan) ChatGPT'ye size Boltzman Machine ilgili detaylı bilgi vermesini isteyin. Sorduğunuz soruyu, ChatGPT'nin çıkarttığı cevabı aşağı koyun. Bu cevapta anlamadığınız terimleri de ChatGPT'ye sorup, sorduğunuz soruları ve ChatGPT'nin verdiği cevapları da sırayla aşağı koyun.

Boltzmann Machine (Boltzmann Makinesi), yapay sinir ağı alanında kullanılan bir tür makine öğrenmesi algoritmasıdır. Boltzmann Machine, unsupervised learning (denetimsiz öğrenme) algoritmalarından biridir.

Boltzmann Machine, yüksek boyutlu verilerdeki gizli yapıları keşfetmek için kullanılır. Bu yapılar, örüntü tanıma, veri sıkıştırma ve doğal dil işleme gibi birçok farklı uygulama için önemlidir. Boltzmann Machine, sinir ağları olarak adlandırılan bir tür modeldir ve düğümler (nöronlar) ve ağırlıklar arasındaki bağlantılar ile tanımlanır.

Bir Boltzmann Machine, iki farklı tipte düğümlerden oluşur: gözlem düğümleri ve gizli düğümler. Gözlem düğümleri, veri özelliklerini temsil ederken, gizli düğümler, verilerdeki gizli yapıları temsil eder. Boltzmann Machine'de, her düğüm belirli bir olasılık dağılımına sahiptir.

Bir Boltzmann Machine'in en önemli özelliği, enerji fonksiyonudur. Bu fonksiyon, ağırlıklar ve düğümler arasındaki bağlantılarla hesaplanır. Boltzmann Machine'in amacı, enerji fonksiyonunun minimumunu bulmaktır. Bu minimum, Boltzmann Machine'in kararını verir.

Boltzmann Machine'in öğrenme süreci, Gibbs örnekleme yöntemi adı verilen bir yöntemle gerçekleştirilir. Bu yöntem, rastgele bir durumdan başlar ve enerji fonksiyonunu minimum yapan bir duruma ulaşana kadar durumları rastgele değiştirir. Bu sayede, Boltzmann Machine, verilerdeki gizli yapıları keşfedebilir.

Boltzmann Machine, özellikle resim tanıma, doğal dil işleme ve ses işleme gibi alanlarda kullanılmaktadır. Bunun yanı sıra, Boltzmann Machine, veri sıkıştırma, boyut indirgeme ve veri tamamlama gibi birçok farklı uygulama için de kullanılabilir.

Sonuç olarak, Boltzmann Machine, yapay sinir ağı alanında kullanılan ve verilerdeki gizli yapıları keşfetmek için kullanılan bir tür makine öğrenmesi algoritmasıdır. Boltzmann Machine'in enerji fonksiyonu ve Gibbs örnekleme yöntemi, algoritmanın öğrenme sürecini yönetir ve verilerdeki gizli yapıların keşfedilmesini sağlar.

2.2 (8 Puan) ChatGPT'nin açıklamalarından ne anladığınızı buraya kendi kelimelerinizle özetleyin. Örneğin "Boltzman Machine nedir? Ne işe yarar? Neden bilmemiz gerekir bunu? Gerçek hayatta kullanılıyor mu?" gibi. Anlamadığınız kısımlar varsa ve ChatGPT'nin açıklamaları yeterli gelmezse internet dahil farklı kaynaklardan araştırıp, bilgilerin doğruluğunu sorgulamakta serbestsiniz. Konuyu doğru anlayıp anlamamanız puana etki edecektir. Kullandığınız kaynakları da belirtin!

Boltzmann Machine de Hopfield ağları gibi denetimsiz öğrenme (unsupervised learning) yöntemlerinden biridir. Derste tahtaya çizdiğiniz gibi giriş nöronlar bağlı oldukları diğer çıkış nöronlarını besliyordu. Boltzmann Machine' de yapay sinir ağı modellerindendir ve nöronlarının birbirine bağlandığı ağdır. İki farklı düğümlerden oluşuyor bunlardan observation nodes yani gözlem düğümleri veri setindeki değişkenlerin özelliklerini temsil ediyor. Mesela metindeki harfler, çizgilerin koordinatları vs. Bu gözlem düğümleri modelin eğitiminde kullanılır. Bu içerdiği özellikleriyle öğrenmeyi kolaylaştırır. Diğer düğüm ise hidden nodes yani gizli düğümler adı üstünde verideki gizli yapıları temsil ediyor. Gizli düğümler veri setindeki farklı değişkenler arasındaki bağımlılıkları yakalayarak özellikleri daha iyi anlamasını sağlar ve yüksek boyutlu verilerde gizli katman ile görüntüdeki nesneleri tanımak için kullanılabilir.

Boltzmann makinesi veri setindeki özellikleri göz önünde bulundurarak düğümlerin durumlarını belirlerken enerji fonksiyonundan yararlanır. Mesela doğal dil işlemede enerji fonksiyonu, cümlelerdeki kelime anlamları arasındaki ilişkileri yakalamak için bu ağ yapısını kullanır.

Boltzmann makinesi, birçok alanda kullanıldığından, günümüzde makine öğrenimi ve yapay zeka teknolojileri için önemli bir araç haline gelmiştir. Gerçek hayatta görüntü işleme, finansal veri analizi gibi yerlerde kullanılabilir.

3 (Toplam 10 Puan) Markov Assumption ve Markov Chain:

3.1 (2 Puan) ChatGPT'ye size Markov assumption ve Markov Chain ilgili detaylı bilgi vermesini isteyin. Sorduğunuz soruyu, ChatGPT'nin çıkarttığı cevabı aşağı koyun. Bu cevapta anlamadığınız terimleri de ChatGPT'ye sorup, sorduğunuz soruları ve ChatGPT'nin verdiği cevapları da sırayla aşağı koyun.

Markov varsayımı, bir sürecin gelecekteki durumu, yalnızca o anki durumuna bağlıdır ve önceki durumlar geçmişte kalmıştır. Bu varsayım, birçok alanda, özellikle de zaman serileri analizinde ve doğal dil işleme alanlarında kullanılır.

Markov zinciri, bir Markov varsayımına dayalı olarak tasarlanan bir olasılık modelidir. Bir Markov zinciri, bir dizi durumdan oluşur ve her bir durum, bir sonraki duruma geçişin olasılığını belirleyen bir olasılık dağılımı ile ilişkilidir.

Markov zinciri, herhangi bir zamanda, o anki duruma bağlı olarak gelecekteki durumun olasılığını hesaplayabilir. Bu hesaplama, bir sonraki durumun olasılığını hesaplamak için mevcut durumu ve geçmişteki tüm durumları kullanmayı gerektirmez, sadece mevcut durumu kullanır. Bu özellik, Markov zincirini özellikle büyük veri kümelerinde kullanışlı hale getirir.

Örneğin, bir hava durumu modeli, bir Markov zinciri olarak tasarlanabilir. Bu zincir, hava durumunun farklı durumlarını temsil eden durumlar dizisinden oluşur. Her bir durum, bir sonraki durumun olasılığını belirleyen bir olasılık dağılımı ile ilişkilidir. Bu olasılık dağılımı, örneğin mevcut hava durumuna ve mevsime bağlı olarak belirlenebilir.

Markov zincirleri, zaman serilerinde sıklıkla kullanılır. Örneğin, bir hisse senedi fiyatının gelecekteki değerlerini tahmin etmek için bir Markov zinciri kullanılabilir. Bu zincir, geçmiş fiyat hareketlerinden oluşan bir durumlar dizisinden oluşur ve her bir durum, bir sonraki fiyat hareketinin olasılığını belirleyen bir olasılık dağılımı ile ilişkilidir.

Markov zincirleri, doğal dil işleme alanında da kullanılır. Örneğin, bir dil modeli, bir Markov zinciri olarak tasarlanabilir. Bu zincir, farklı kelimelerin dizilerinden oluşan bir durumlar dizisinden oluşur ve her bir durum, bir sonraki kelimenin olasılığını belirleyen bir olasılık dağılımı ile ilişkilidir. Bu model, bir metnin dilbilgisel özelliklerini yakalamak için kullanılabilir.

3.2 (8 Puan) ChatGPT'nin açıklamalarından ne anladığınızı buraya kendi kelimelerinizle özetleyin. Örneğin "Markov assumption ve Markov Chain nedir? Ne işe yarar? Neden bilmemiz gerekir bunu? Gerçek hayatta kullanılıyor mu?" gibi. Anlamadığınız kısımlar varsa ve ChatGPT'nin açıklamaları yeterli gelmezse internet dahil farklı kaynaklardan araştırıp, bilgilerin doğruluğunu sorgulamakta serbestsiniz. Konuyu doğru anlayıp anlamamanız puana etki edecektir. Kullandığınız kaynakları da belirtin!

Markov varsayımı mevcut state yani durumunun olasılığı ve kendinden bir önceki statelere bağlıdır. Mesela state x'in olasılığını hesaplarken x-1' inci state' i kullanırız. Markov zinciri ile statelerin durumlarına göre gelecekteki state' in olasılığını hesaplayabilir. Çok geçmişe gitmediğinden onun için şu anki durum ve gelecek önemli olduğundan büyük verilerde çalıştığımızda işlerimiz karmaşıklaşmaz,

çok veriyle karşılaştırma yapmak zorunda kalmayız. Gerçek hayatta sanırım bir çok yerde Markov yaklaşımını ve zincirini adını bilmeden kullanıyoruz. En güzel örneği hava durumu geldi bana. Çünkü tanımları güzel karşılıyor. Mesela yarının hava durumunu kestirmek için çokta eski zamanlara gidip derecelerine bakmamıza gerek yok. Haliyle mevsimler değişiyor. Bizim için önceki gün ve bugünü bilmek yarını tahmin etmemizde daha iyi sonuçlar doğurabilir. Bunun dışında daha bir çok alanda kullanılıyor bu kavramlar. Örneğin hisselerin değişim tahmini, robotların hareketleri…

4 (Toplam 20 Puan) Feed Forward:

- Forward propagation için, input olarak ¸şu X matrisini verin (tensöre çevirmeyi unutmayın):
  X = $\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$ Satırlar veriler (sample'lar), kolonlar ¨oznitelikler (feature'lar).
- Bir adet hidden layer olsun ve içinde tanh aktivasyon fonksiyonu olsun
- Hidden layer'da 50 nöron olsun
- Bir adet output layer olsun, tek nöronu olsun ve içinde sigmoid aktivasyon fonksiyonu olsun

Tanh fonksiyonu:

$$f(x) = \frac{exp(x) - exp(-x)}{exp(x) + exp(-x)}$$

Sigmoid fonksiyonu:

$$f(x) = \frac{1}{1 + exp(-x)}$$

Pytorch kütüphanesi ile, ama kütüphanenin hazır aktivasyon fonksiyonlarını kullanmadan, formülünü verdiğim iki aktivasyon fonksiyonunun kodunu ikinci haftada yaptığımız gibi kendiniz yazarak bu yapay sinir ağını oluşturun ve aşağıdaki üç soruya cevap verin.

4.1 (10 Puan) Yukarıdaki yapay sinir ağını çalıştırmadan önce pytorch için Seed değerini 1 olarak set edin, kodu aşağıdaki kod bloğuna ve altına da sonucu yapıştırın:

```
import torch
import torch.nn as nn

class MyNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super(MyNet, self).__init__()
        self.hidden_layer = nn.Linear(input_size, hidden_size)
        self.output_layer = nn.Linear(hidden_size, output_size)

    def tanh(self, x):
        return torch.tanh(x)
```

```
    def sigmoid(self, x):
        return 1 / (1 + torch.exp(-x))

    def forward(self, x):
        h = self.tanh(self.hidden_layer(x))
        y = self.sigmoid(self.output_layer(h))
        return y

# Seed: aynı rastgele sayılar üretmesini sağlamak
torch.manual_seed(1)
```

```
# Input değerleri
X = torch.tensor([[1, 2, 3],
                  [4, 5, 6]], dtype=torch.float32)

# Weight ve bias
input_size = 3
hidden_size = 50
output_size = 1

# Model oluşturma
model = MyNet(input_size, hidden_size, output_size)

# Print model parameters
print("Modelin parametreleri aşağıdaki şekildedir:")
for name, param in model.named_parameters():
    print(name, param.size())

# girdi tensörü X ile ileri doğru yayılım (forward propagation) işlemi
uygular ve çıktı tensörü y'yi hesaplar.
y = model(X)

# Print output
print("Çıktı katmanı :")
print(y)
```

```
Modelin parametreleri aşağıdaki şekildedir:
hidden_layer.weight torch.Size([50, 3])
hidden_layer.bias torch.Size([50])
output_layer.weight torch.Size([1, 50])
output_layer.bias torch.Size([1])
Çıktı katmanı :
tensor([[0.4892],
        [0.5566]], grad_fn=<MulBackward0>)
```

4.2 (5 Puan) Yukarıdaki yapay sinir ağını çalıştırmadan önce Seed değerini öğrenci numaranız olarak değiştirip, kodu aşağıdaki kod bloğuna ve altına da sonucu yapıştırın:

```
import torch
import torch.nn as nn

class MyNet(nn.Module):
    def_init_(self, input_size, hidden_size, output_size):
        super(MyNet, self).__init__()
        self.hidden_layer = nn.Linear(input_size, hidden_size)
        self.output_layer = nn.Linear(hidden_size, output_size)
    def tanh(self, x):
        return torch.tanh(x)

    def sigmoid(self, x):
        return 1 / (1 + torch.exp(-x))

    def forward(self, x):
        h = self.tanh(self.hidden_layer(x))
        y = self.sigmoid(self.output_layer(h))
        return y

# Seed: aynı rastgele sayılar üretmesini sağlamak
torch.manual_seed(190401083)

# Input değerleri
X = torch.tensor([[1, 2, 3],
                  [4, 5, 6]], dtype=torch.float32)

# Weight ve bias
input_size = 3
hidden_size = 50
output_size = 1

# Model oluşturma
model = MyNet(input_size, hidden_size, output_size)

# Print model parameters
print("Modelin parametreleri aşağıdaki şekildedir:")
for name, param in model.named_parameters():
    print(name, param.size())

# girdi tensörü X ile ileri doğru yayılım (forward propagation) işlemi
uygular ve çıktı tensörü y'yi hesaplar.
y = model(X)

# Print output
print("Çıktı katmanı :")
print(y)
```

```
Modelin parametreleri aşağıdaki şekildedir:
hidden_layer.weight torch.Size([50, 3])
hidden_layer.bias torch.Size([50])
output_layer.weight torch.Size([1, 50])
output_layer.bias torch.Size([1])
Çıktı katmanı :
tensor([[0.6837],
        [0.6456]], grad_fn=<MulBackward0>)
```

4.3 (5 Puan) Kodlarınızın ve sonuçlarınızın olduğu jupyter notebook'un Github repository'sindeki linkini aşağıdaki url kısmının içine yapıştırın. İlk sayfada belirttiğim gün ve saate kadar halka acık (public) olmasın:

https://github.com/Esra08/Yapay-Sinir-Aglari-Project/blob/main/YSA_VizeProject_Question4.ipynb

5 (Toplam 40 Puan) Multilayer Perceptron (MLP):

Bu bolümdeki sorularda benim vize ile beraber paylaştığım Prensesi İyileştir (Cure The Princess) Veri Seti parçaları kullanılacak. Hikaye söyle (soruyu çözmek icin hikaye kısmını okumak zorunda değilsiniz):

Iki hidden layer'lı bir Multilayer Perceptron (MLP) oluşturun besinci ve altıncı haftalarda yaptığımız gibi. Hazır aktivasyon fonksiyonlarını kullanmak serbest. İlk hidden layer'da 100, ikinci hidden layer'da 50 nöron olsun. Hidden layer'larda ReLU, output layer'da sigmoid aktivasyonu olsun.

Output layer'da kaç nöron olacağını veri setinden bakıp bulacaksınız. Elbette bu veriye uygun Cross Entropy loss yöntemini uygulayacaksınız. Optimizasyon için Stochastic Gradient Descent yeterli. Epoch sayınızı ve learning rate'i validasyon seti üzerinde denemeler yaparak (loss'lara overfit var mı diye bakarak) kendiniz belirleyeceksiniz. Batch size'ı 16 seçebilirsiniz.

5.1 (10 Puan) Bu MLP'nin pytorch ile yazılmış class'ının kodunu aşağı kod bloğuna yapıştırın:

```
# Multilayer perceptron (MLP) modelini tanımlamak
class MLP(nn.Module):
    def_init_(self, input_size, hidden_size1, hidden_size2, output_s
ize):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
```

```
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, output_size)
        self.sigmoid = nn.Sigmoid()

    def  forward(self,  x):
        out  =  self.fc1(x)
        out = self.relu1(out)
        out = self.fc2(out)
        out = self.relu2(out)
        out = self.fc3(out)
        out = self.sigmoid(out)
        return out
```

5.2 (10 Puan) SEED=öğrenci numaranız set ettikten sonra altıncı haftada yazdığımız gibi training batch'lerinden eğitim loss'ları, validation batch'lerinden validasyon loss değerlerini hesaplayan kodu aşağıdaki kod bloğuna yapıştırın ve çıkan figürü de alta ekleyin.

```
%load_ext autotime

SEED = 190401083
torch.manual_seed(SEED)

class MLP(nn.Module):
    def_init_(self, input_size, hidden_size1, hidden_size2, output_s
ize):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()
        self.fc3 = nn.Linear(hidden_size2, output_size)
        self.sigmoid = nn.Sigmoid()

    def  forward(self,  x):
        out  =  self.fc1(x)
        out = self.relu1(out)
        out = self.fc2(out)
        out = self.relu2(out)
        out = self.fc3(out)
        out = self.sigmoid(out)
        return out

input_size = 13
hidden_size1 = 100
hidden_size2 = 50
output_size = 1
```

```
model = MLP(input_size, hidden_size1, hidden_size2, output_size)

#PyTorch modelinin eğitimi için kayıp fonksiyonunu ve optimize ediciyi
belirlemek
criterion = nn.BCELoss()
optimizer = optim.SGD(model.parameters(), lr=0.001)

# Eğitim ve doğrulama kayıp değerlerini takip etmek ve en düşük doğrula
ma kayıp değerlerini belirlemek için gerekli değişkenleri tanımlar.
list_train_loss, list_val_loss = [], []
best_val_loss = None
patience_counter = 0

# Belirtilen sayıda epoch için modelin eğitimini gerçekleştiriyor.
for epoch in range(num_epochs):

    train_loss = 0.0
    train_count = 0.0
    for inputs, labels in dataloader_train:
        optimizer.zero_grad()   #gradyanları sıfırlıyoruz
        outputs = model(inputs)   #model(inputs) ile girdileri modele v
eriyoruz ve çıktıları alıyoruz
        loss = criterion(outputs, labels)  #çıktılar ve etiketler arası
ndaki kaybı hesaplıyoruz
        loss.backward()  #gradyanları hesaplıyoruz
        optimizer.step()  #parametreleri güncelliyoruz

        # Her batch için, train_loss değişkenine batch'teki loss değeri
 ekleniyor ve train_count değişkeni de bir artırılıyor. Bu sayede, epoc
h boyunca ortalama train loss hesaplanabilecek
        train_count += 1.0
        train_loss += loss.item()

    # Her epoch sonrasında eğitim seti ve doğrulama seti için kayıp değ
erlerinin hesaplanması
    val_loss = 0.0
    with torch.no_grad():
        model.eval()
        for inputs, labels in dataloader_valid:
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
    model.train()

    train_loss /= train_count
    val_loss /= len(dataloader_valid)

    print("Epoch", epoch, "Training loss", train_loss,"Validation Loss
:",val_loss)
```

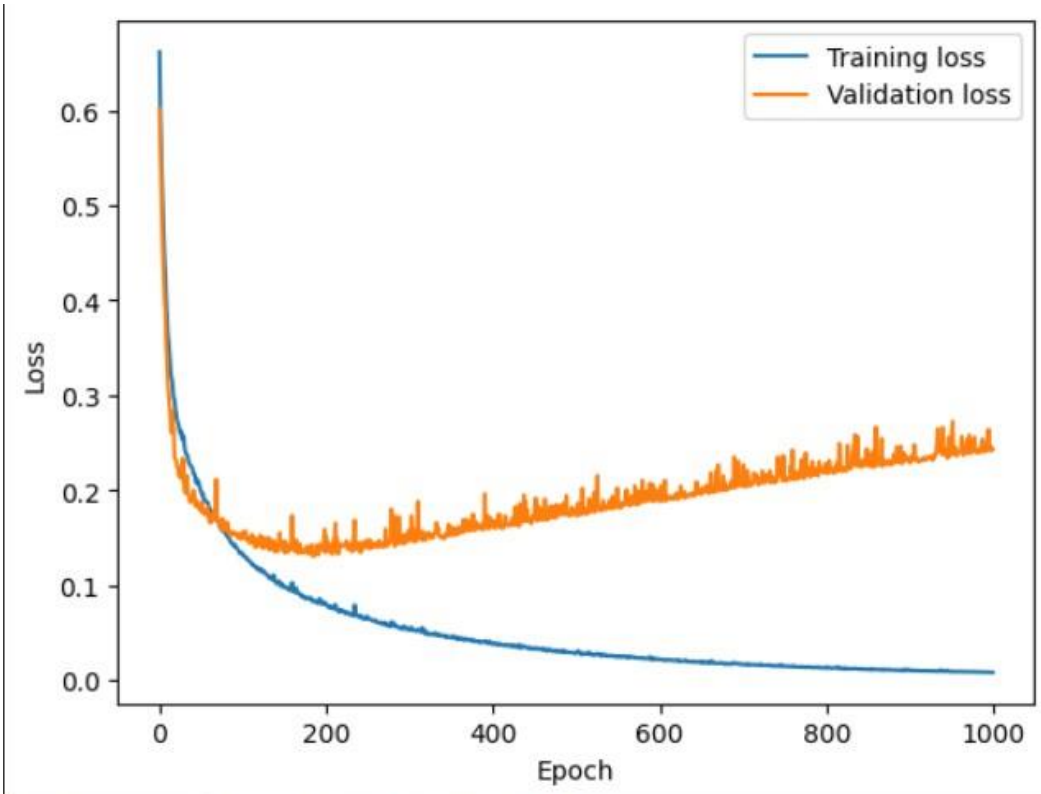
```
    list_train_loss.append(train_loss)
    list_val_loss.append(val_loss)

    #Early stopping yapıldı. Model, her epoch sonunda doğrulama veriler
indeki kaybın en düşük olduğu durumda kaydedilir ve belirli bir sayıda
epoch boyunca (patience) doğrulama kaybı azalmazsa eğitim sonlandırılır
.
    val_score = val_loss
    if best_val_loss is None:
        best_val_loss = val_score
        torch.save(model.state_dict(), "checkpoint.pt")
    elif best_val_loss < val_score:
        #patience:mevcut doğruluk değeri ile en iyi kaydedilen doğruluk
 değeri arasındaki fark
        patience_counter += 1
        print("Earlystopping Patience Değeri:", patience_counter)
        if patience_counter == patience:
            break
    else:
        best_val_loss = val_score
        torch.save(model.state_dict(), "checkpoint.pt")
        patience_counter = 0

# Bu kod bloğu eğitim sürecinde kaydedilen eğitim ve doğrulama (validat
ion) kayıplarının grafiklerini çizdirir.
# X ekseni epoch sayısını, y ekseni kayıp (loss) değerini gösterir.
# Bu grafikler, modelin eğitim ve doğrulama performansını izlemek ve aş
ırı uydurmayı (overfitting) tespit etmek için kullanılabilir.
plt.plot(list_train_loss, label="Training loss")
plt.plot(list_val_loss, label="Validation loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

5.3 (10 Puan) SEED= öğrenci numaranız set ettikten sonra altıncı haftada ödev olarak verdiğim gibi earlystopping'deki en iyi modeli kullanarak, Prensesi İyileştir test setinden accuracy, F1, precision ve recall değerlerini hesaplayan kodu yazın ve sonucu da aşağı yapıştırın. %80'den fazla başarı bekliyorum test setinden. Daha düşükse başarı oranınız, nerede hata yaptığınızı bulmaya çalısın. %90'dan fazla başarı almak mümkün (ben denedim).

```
model = MLP(input_size, hidden_size1, hidden_size2, output_size)
model.load_state_dict(torch.load('checkpoint.pt'))
model.eval()
predicts =[]
real_labels = list()
with torch.no_grad():
```

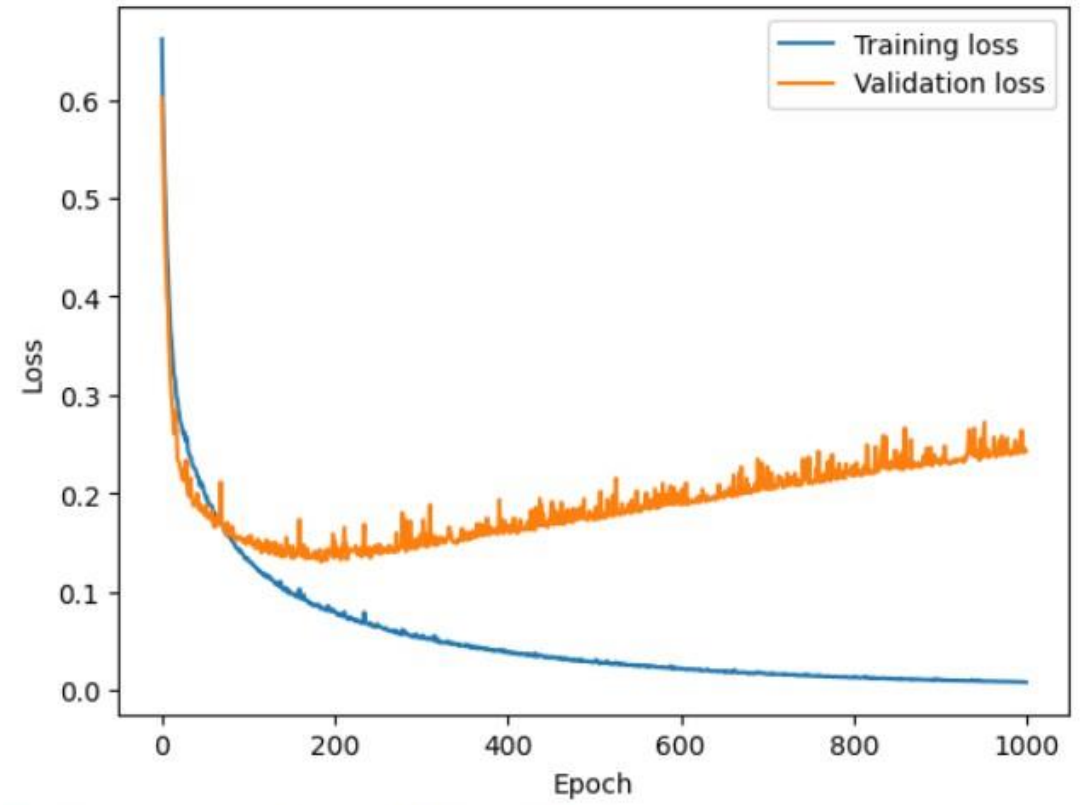
```
    for inputs,label in dataloader_test:
        outputs = model(inputs)
        for out in outputs:

            predict = round(float(out.data))
            predicts.append(predict)
        real_labels.extend(label.tolist())

from sklearn.metrics import f1_score,accuracy_score,classification_repo
rt
print("Modelin doğruluk yüzdesi: {}".format(accuracy_score(real_labels,
predicts)))
print(classification_report(real_labels,predicts))
```

***5.2 ve 5.3'ün CPU çıktısı:**




```
Modelin doğruluk yüzdesi: 0.9481865284974094
              precision    recall  f1-score   support

         0.0       0.93      0.96      0.95       384
         1.0       0.96      0.93      0.95       388

    accuracy                           0.95       772
   macro avg       0.95      0.95      0.95       772
weighted avg       0.95      0.95      0.95       772

time: 1min 17s (started: 2023-04-14 07:07:40 +00:00)
```

***5.2 ve 5.3'ün GPU çıktısı:**




```
Modelin doğruluk yüzdesi: 0.9481865284974094
              precision    recall  f1-score   support

         0.0       0.93      0.96      0.95       384
         1.0       0.96      0.93      0.95       388

    accuracy                           0.95       772
   macro avg       0.95      0.95      0.95       772
weighted avg       0.95      0.95      0.95       772

time: 1min 5s (started: 2023-04-14 08:04:29 +00:00)
```

***GPU ile çalışmak için kodlar:**

```
%load_ext autotime
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
model = MLP(input_size, hidden_size1, hidden_size2, output_size)
optimizer = optim.SGD(model.parameters(), lr=0.001)
criterion = nn.BCELoss()


model.to(device)

list_train_loss, list_val_loss = [], []
best_val_loss = None
patience_counter = 0
```

```
for epoch in range(num_epochs):

    train_loss = 0.0
    train_count = 0.0
    for inputs, labels in dataloader_train:
        inputs, labels = inputs.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward(torch.ones_like(loss))
        optimizer.step()

        train_count += 1.0
        train_loss += loss.item()

    val_loss = 0.0
    with torch.no_grad():
        model.eval()
        for inputs, labels in dataloader_valid:
            inputs, labels = inputs.to(device), labels.to(device)
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()
    model.train()

    train_loss /= train_count
    val_loss /= len(dataloader_valid)

    print("Epoch", epoch, "Training loss", train_loss,"Validation Loss
:",val_loss)
    list_train_loss.append(train_loss)
    list_val_loss.append(val_loss)

plt.plot(list_train_loss, label="Training loss")
plt.plot(list_val_loss, label="Validation loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()
```

5.4 (5 Puan) Tüm kodların CPU'da çalışması ne kadar sürüyor hesaplayın. Sonra to device yöntemini kullanarak modeli ve verileri GPU'ya atıp kodu bir de böyle çalıştırın ve ne kadar sürdüğünü hesaplayın. Süreleri aşağıdaki tabloya koyun. GPU için Google Colab ya da Kaggle'ı kullanabilirsiniz, iki ortam da her hafta saatlerce GPU hakkı veriyor.

Tablo 1: Buraya bir açıklama yazın

Ortam	Süre (saniye)
CPU	1min 17s
GPU	1min 5s

5.5 (3 Puan) Modelin eğitim setine overfit etmesi için elinizden geldiği kadar kodu gereken şekilde değiştirin, validasyon loss'unun açıkça yükselmeye başladığı, training ve validation loss'ları içeren figürü aşağı koyun ve overfit için yaptığınız değişiklikleri aşağı yazın. Overfit, tam bir çanak gibi olmalı ve yükselmeli. Ona göre parametrelerle oynayın.

Batch size azaltmak, learning rate arttırmak, nöron sayısıyla oynamak gerekiyor. Uğraştım ama dediğiniz şekle ulaşamadığım için ekran görüntüsü eklemedim.

5.6 (2 Puan) Besinci soruya ait tüm kodların ve cevapların olduğu jupyter notebook'un Github linkini aşağıdaki url'e koyun.

https://github.com/Esra08/Yapay-Sinir-Aglari-Project/blob/main/YSA_VizeProject_AllQuestions.ipynb

6 (Toplam 10 Puan)

Bir önceki sorudaki Prensesi İyileştir problemindeki yapay sinir ağınıza seçtiğiniz herhangi iki farklı regülarizasyon yöntemi ekleyin ve aşağıdaki soruları cevaplayın.

6.1 (2 puan) Kodlarda regülarizasyon eklediğiniz kısımları aşağı koyun:

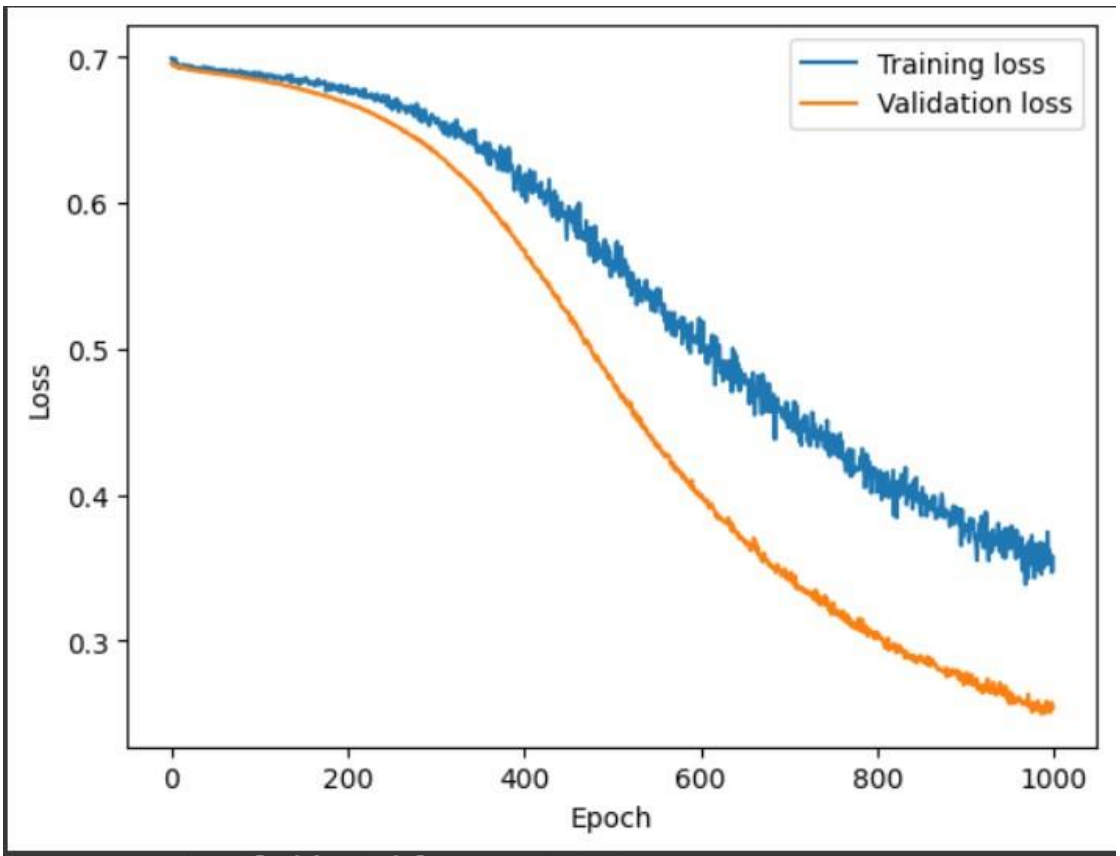
```
%load_ext autotime

class MLP(nn.Module):
    def_init_(self, input_size, hidden_size1, hidden_size2, output_s
ize, weight_decay=0.01,dropout_prob=0.5):
        super(MLP, self).__init__()
        self.fc1 = nn.Linear(input_size, hidden_size1)
        self.relu1 = nn.ReLU()
        self.dropout1 = nn.Dropout(dropout_prob)
        self.fc2 = nn.Linear(hidden_size1, hidden_size2)
        self.relu2 = nn.ReLU()
        self.dropout2 = nn.Dropout(dropout_prob)
        self.fc3 = nn.Linear(hidden_size2, output_size)
        self.sigmoid = nn.Sigmoid()
        self.weight_decay = weight_decay

    def  forward(self,  x):
        out  =  self.fc1(x)
        out = self.relu1(out)
        out = self.dropout1(out)
        out = self.fc2(out)
```

```
        out = self.relu2(out)
        out = self.dropout2(out)
        out = self.fc3(out)
        out = self.sigmoid(out)
        return out

    def l2_regularization(self):
        total_weight = 0.0

        for param in self.parameters():
            total_weight += torch.sum(torch.square(param))

        return self.weight_decay * total_weight

    def loss(self, output, target):
        ce_loss = F.binary_cross_entropy(output, target)

        # weight decay ile L2 regülarizasyonunun eklenmesi
        l2_reg = self.l2_regularization()

        # toplam loss değerinin döndürülmesi
        return ce_loss + l2_reg


input_size = 13
hidden_size1 = 100
hidden_size2 = 50
output_size = 1

model = MLP(input_size, hidden_size1, hidden_size2, output_size)

criterion = nn.BCELoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, weight_decay= 1e-6)

list_train_loss, list_val_loss = [], []
best_val_loss = None

patience_counter = 0
for epoch in range(num_epochs):

    train_loss = 0.0
    train_count = 0.0
    for inputs, labels in dataloader_train:

        optimizer.zero_grad()
        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
```

```
        train_count += 1.0
        train_loss += loss.item()

    val_loss = 0.0
    with torch.no_grad():
        model.eval()
        for inputs, labels in dataloader_valid:
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            val_loss += loss.item()

    model.train()
    train_loss /= train_count
    val_loss /= len(dataloader_valid)
    print("Epoch", epoch, "Training loss", train_loss,"Validation Loss
:",val_loss)

    list_train_loss.append(train_loss)
    list_val_loss.append(val_loss)

    val_score = val_loss
    if best_val_loss is None:
        best_val_loss = val_score
        torch.save(model.state_dict(), "checkpoint.pt")
    elif best_val_loss < val_score:
        patience_counter += 1
        print("Earlystopping Patience Counter:",patience_counter)
        if patience_counter == patience:
            break
    else:
        best_val_loss = val_score
        torch.save(model.state_dict(), "checkpoint.pt")
        patience_counter = 0

plt.plot(list_train_loss, label="Training loss")
plt.plot(list_val_loss, label="Validation loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.legend()
plt.show()

model = MLP(input_size, hidden_size1, hidden_size2, output_size)
model.load_state_dict(torch.load('checkpoint.pt'))
model.eval()
predicts =[]
real_labels = list()
with torch.no_grad():
    for inputs,label in dataloader_test:
```

```
        outputs = model(inputs)
        for out in outputs:

            predict = round(float(out.data))
            predicts.append(predict)
        real_labels.extend(label.tolist())

from sklearn.metrics import f1_score,accuracy_score,classification_repo
rt
print("Accuracy score of this model: {}".format(accuracy_score(real_lab
els,predicts)))
print(classification_report(real_labels,predicts))
```

6.2 (2 puan) Test setinden yeni accuracy, F1, precision ve recall değerlerini hesaplayıp aşağı koyun:




```
Accuracy score of this model: 0.8886010362694301
              precision    recall  f1-score   support

         0.0       0.88      0.89      0.89       384
         1.0       0.89      0.88      0.89       388

    accuracy                           0.89       772
   macro avg       0.89      0.89      0.89       772
weighted avg       0.89      0.89      0.89       772
```

6.3 (5 puan) Regülarizasyon yöntemi seçimlerinizin sebeplerini ve sonuçlara etkisini yorumlayın:

Regularizasyon yöntemleri, makine öğrenmesi ve derin öğrenme modellerinde aşırı öğrenmeyi önlemek ve genelleme performansını artırmak için kullanılan tekniklerdir. Bu tekniklerden weight decay ve dropout ‘ u seçtim. Çünkü diğer tekniklere göre basit ve kolay uygulanabilir olduklarını, etkili sonuçlar verdiklerini, genellikle az hesaplama maliyeti gerektirdiklerini ve modellerin açıklanabilirliği üzerinde daha az etkiye sahip olduğunu düşündüm. Sonuçlara etkileri de weight decay için ağırlıkların büyüklüğüne bağlı olarak farklı değerler seçmek gerekebilir, dropout için daha yavaş eğitim sürecine neden olabilir.

6.4 (1 puan) Sonucun github linkini aşağıya koyun:

https://github.com/Esra08/Yapay-Sinir-Aglari-Project/blob/main/YSA_VizeProject_AllQuestions.ipynb