İstanbul Teknik Üniversitesi
Elektrik ve Elektronik Fakültesi
Elektronik ve Haberleşme Mühendisliği

Yapay Sinir Ağları

Ödev-2

Dersi Veren Öğretim Üyesi

Prof. Dr. Neslihan Serap ŞENGÖR

Öğrenci Grubu 3

Yunus Örük

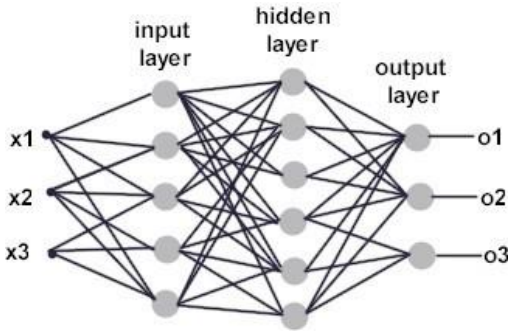
040170083

Yusuf Demirel

040180033

MULTILAYER PERCEPTRON (MLP)

Çok katmanlı algılayıcılar XOR problemine çözüm üretmek üzere yapılan çalışmalar sonucunda ortaya çıkmıştır. Özellikle sınıflandırma problemlerinde etkin olarak çalışmaktadır. Tek bir katmana sahip oldukları durumlarda "vanilla" sinir ağları olarak adlandırılmaktadırlar.



XOR

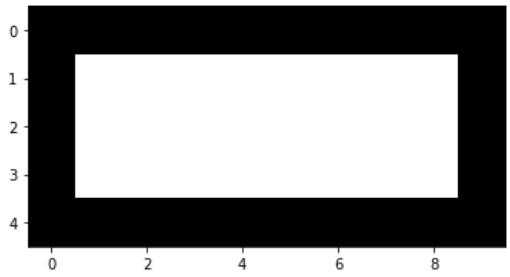
Input x_1	Input x_2	Output
0	0	0
0	1	1
1	0	1
1	1	0

Çok katmanlı algılayıcılar en az üç katmandan oluşmaktadırlar. Giriş katmanı, gizli katman ve çıkış katmanı. Giriş düğümleri haricinde diğer katmanlardaki düğümler doğrusal olmayan bir aktivasyon fonksiyonu kullanmaktadır. Her katman ardından gelen katmana bağlanmaktadır. Çıkış katmanı veriyi işleyerek ağın çıkışını belirler. Çoklu algılayıcılar eğitim için backpropagation adı verilen denetimli bir öğrenme tekniği kullanır. Çok katmanlı algılayıcıları doğrusal algılayıcılardan ayıran özellik çok katmanlı olmaları ve doğrusal olmayan aktivasyon fonksiyonu kullanmalarıdır. Doğrusal olarak sınıflandırılmayan veriler bu şekilde sınıflandırılabilmektedir.

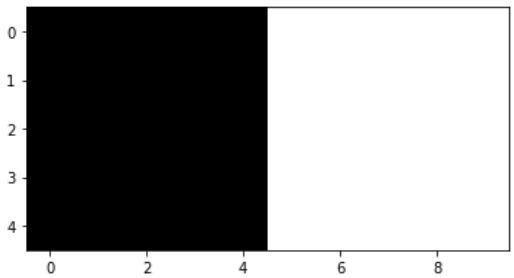
Soru 1.1

A) Öncelikle eğitim kümesi ödev dosyası içinde verilen python kodu fonksiyonlaştırılarak oluşturuldu. Eğitim kümesi için 4 sınıf ve her sınıf için 4 adet gri gürültü eklenmiş örüntü ve 4 adet pixel hatalı olan örüntüler belirlendi. Her bir sınıf için 8 örüntü ve toplamda ise 32 adet örüntü eğitim kümesi için belirlenmiş oldu. Test kümesi için her bir sınıftan 2 adet gri gürültülü ve 2 adet pixel hatalı örüntü seçildi. Böylece her sınıf için 4 ve toplamda 16 adet veriden oluşan test kümesi oluşturuldu.

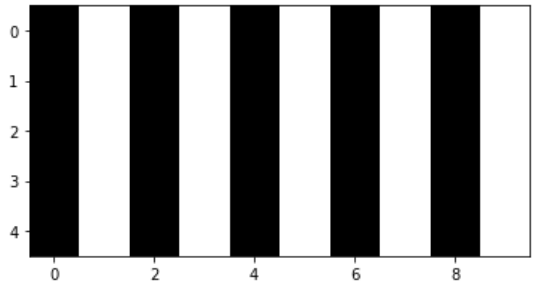
Oluşturulan 4 adet temiz örüntü



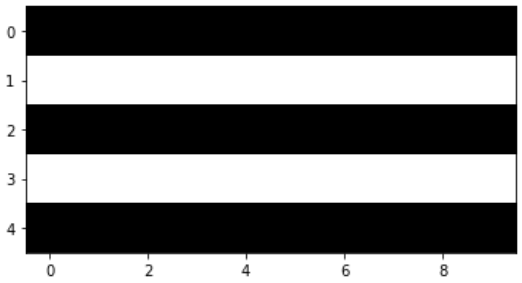
Hamming katsayısı: 1000



Hamming katsayısı: 0100

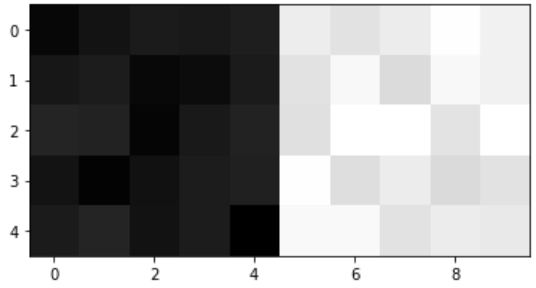


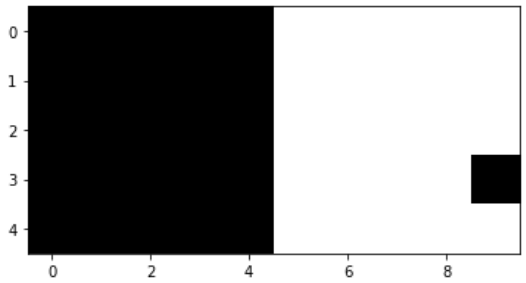
Hamming katsayısı: 0010



Hamming katsayısı: 0001

Oluşturulan örüntülerin gri gürültü ve pixel hatalarına örnekler

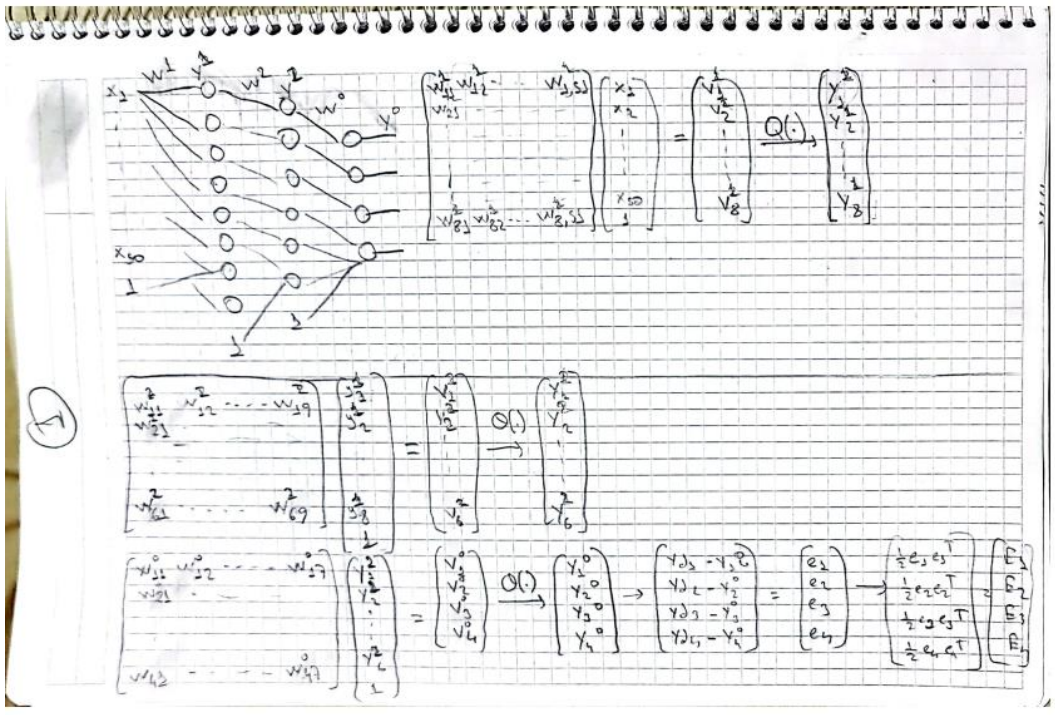




B) Verilerimizi kod kullanarak eğitmeden önce algoritmayı daha iyi anlayabilmek ve daha rahat yazabilmek için kodun kullanacağı algoritmayı kâğıt üzerinde modelledik.

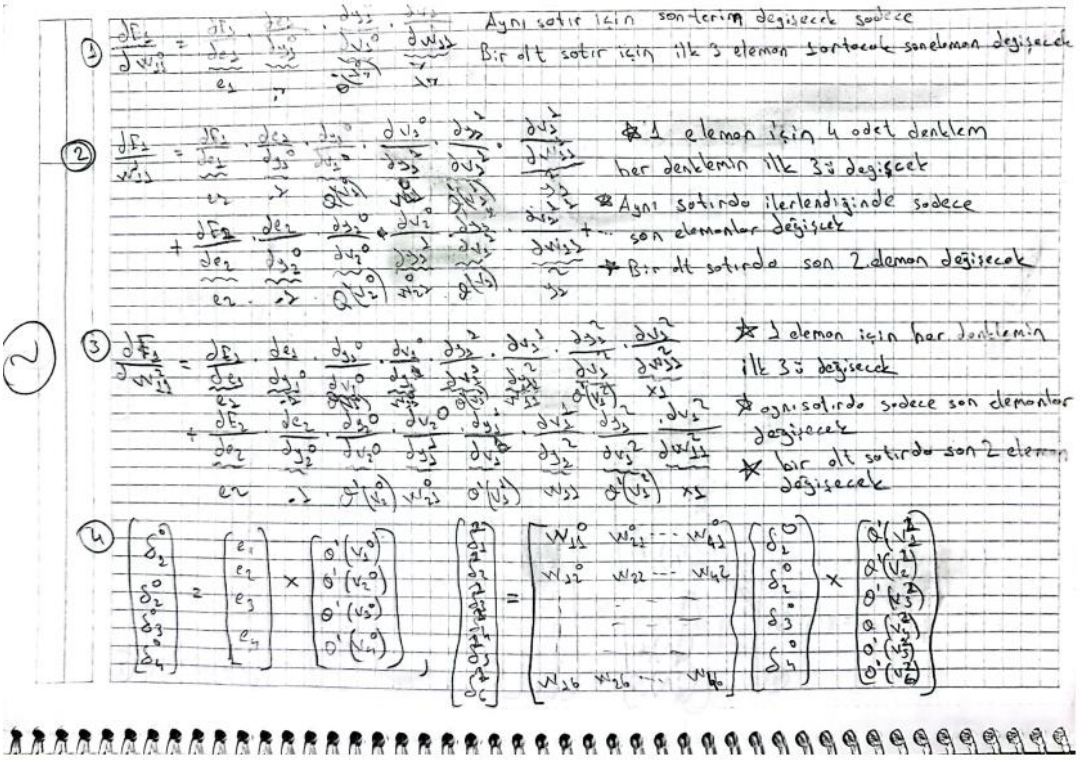
Elimizdeki veriler 5*10'luk boyutta resimler olduğu için giriş vektörü 50+1(bias) olacak şekilde 51 elemanlı bir vektör olarak belirledik. Katman sayımızı 3 ve katmanlardaki nöronları sırasıyla 8 6 ve 4 olarak seçtik. Giriş verilerimiz 0 ve 1 arasında olduğu için aktivasyon fonksiyonu da olarak sigmoid olarak seçtik. Üstteki tasarım seçenekleri belirlendikten sonra algoritmanın ileri yol hesabı şekilde görüldüğü gibi yapıldı.

Algoritma ileri yol hesabı



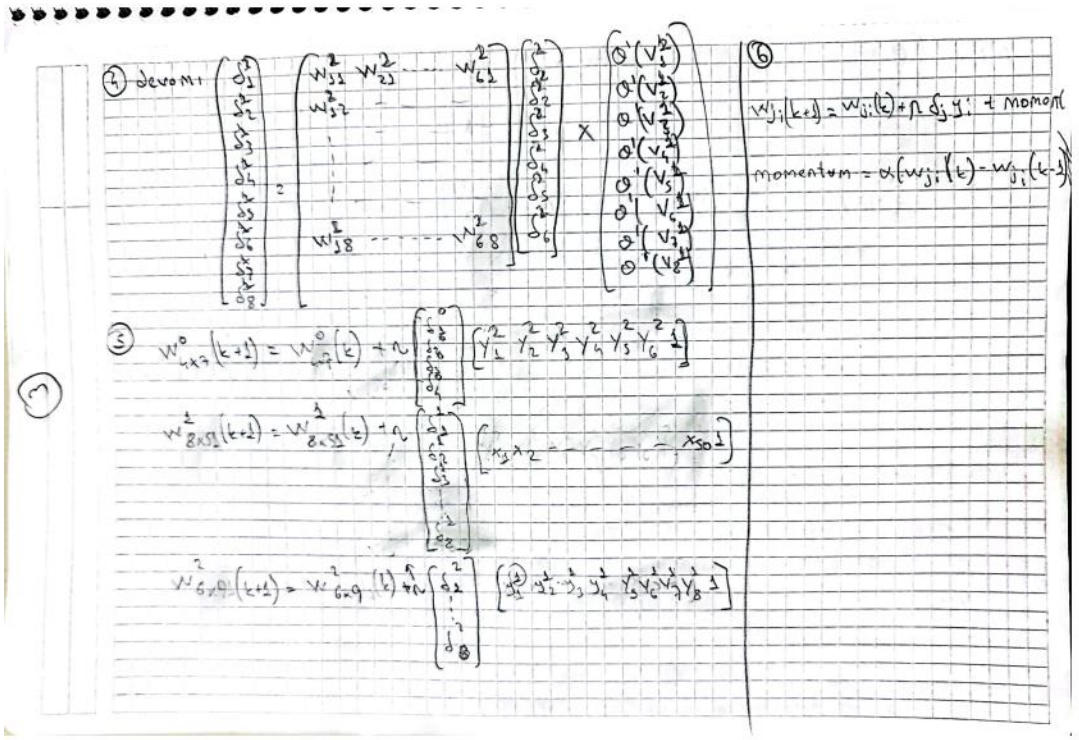
Ardından verilerimizi doğru olarak sınıflandıracak ağırlık değerlerini bulabilmek adına backward propagation yöntemi kullanarak yerel gradyent hesabını yaptık.

Yerel gradyent hesapları



Yerel gradyentler hesaplandıktan sonra ağırlık güncellemelerini gradyentlerin yardımıyla hesapladık. Son olarak ise ağırlık güncellemelerimize momentum terimini de ekledik ve böylece kodu yazmaya hazır hale geldik.

Ağırlık güncellemek için gerekli olan formüller



Durdurma koşulunu 1 epoktaki her bir iterasyondaki hataların ortalamasının 0.01'den küçük gelmesi olarak seçtik. Öğrenme hızı ve momentum terimi katsayılarını her bir katman için farklı değerler verecek şekilde ayarladık. Verilerimizi ağa vermeden önce ise her bir 50+1'lik verimize 4 basamaklı binary Hamming katsayıları atadık.

Katmanlardaki momentum ve learning rate değerleri

```
momentum1 = 0.3          #Eğitim ağının ilk katmanındaki momentum değeri
momentum2 = 0.6          #Eğitim ağının 2. katmanındaki momentum değeri
momentum0 = 0.9          #Eğitim ağının son katmanındaki momentum değeri
learning_rate1 = 0.15    #Eğitim ağının ilk katmanındaki learning değeri
learning_rate2 = 0.30    #Eğitim ağının 2. katmanındaki learning değeri
learning_rate0 = 0.45    #Eğitim ağının son katmanındaki learning değeri
```

```
w1 = np.ones((katman1_norön,51))*0.1                #1. katman ilk ağırlık değerleri
w2 = np.ones((katman2_norön,katman1_norön+1))*0.1   #2. katman ilk ağırlık değerleri
w0 = np.ones((4,katman2_norön+1))*0.1               #3. katman ilk ağırlık değerleri

"Momentum terimi için oluşturulmuş w(k-1) terimleri"
w1_older = np.random.random_sample((katman1_norön,51))*0.1+0.01                #Mome
w2_older = np.random.random_sample((katman2_norön,katman1_norön+1))*0.1+0.01
w0_older = np.random.random_sample((4,katman2_norön+1))*0.1+0.01
```

Eğitime başlamadan önce seçilen ilk ağırlık değerleri (büyüklükleri 0.1) ve momentum teriminin ilk elemanı için rastgele seçilmiş 0.01 ve 0.11 arasındaki ağırlık değerleri bu şekilde belirlendi.

C)

Katmanlardaki Nöron sayılarının etkileri

1. katman Nöron s.	2. katman Nöron s.	Eğitmek için gereken epok sayısı	Ortalama hata
2	2	Tam eğitim yok	0.5
4	4	354	0.0099
4	2	Tam eğitim yok	0.324
2	4	409	0.0098
3	3	372	0.0098
4	3	430	0.0098
8	6	314	0.0098
10	10	286	0.0097
30	20	276	0.0097
30	30	304	0.0096
50	50	Tam eğitim yok	15.99
40	40	Tam eğitim yok	15.99
35	35	489	0.0099

Tabloda görüldüğü üzere katmanlardaki nöron sayıları belirli bir miktara gelesiye kadar sinir ağı eğitim yapamıyor. Ayrıca 4,2 nöronlu ve 2,4 nöronlu örneklere baktığımızda ise her bir katmandaki nöron sayılarının da etkin olduğunu gözlemleyebiliyoruz. Belirli bir nöron sayısı miktarından sonra ise sinir ağı doygunluğa ulaşıyor ve yerel gradyentler 0'a yakınsıyor. Dolayısıyla eğitim gerçekleşmiyor.

Learning rate ve momentum değeri etkileri (Katman sayıları 8,6,4 şeklindedir)

momentum1	momentum2	momentum0	l_rate1	l_rate2	l_rate0	Epok sys
0.3	0.6	0.9	0.15	0.30	0.45	287
0.9	0.6	0.3	0.45	0.3	0.15	-
0.8	0.5	0.2	0.45	0.3	0.15	-
0.7	0.4	0.1	0.45	0.3	0.15	-
0.9	0.6	0.3	0.15	0.3	0.45	-
0.7	0.4	0.1	0.15	0.3	0.45	-
0.5	0.5	0.5	0.15	0.3	0.45	-
0.5	0.6	0.7	0.15	0.3	0.45	-
0.4	0.5	0.6	0.15	0.3	0.45	-
0.6	0.7	0.8	0.15	0.3	0.45	371
0.1	0.5	0.9	0.15	0.3	0.45	348
0.1	0.1	0.1	0.15	0.3	0.45	-
0.3	0.3	0.3	0.15	0.3	0.45	-
0.4	0.4	0.4	0.15	0.3	0.45	-
0.5	0.7	0.9	0.15	0.3	0.45	238
0.7	0.8	0.9	0.15	0.3	0.45	232
0.9	0.9	0.9	0.15	0.3	0.45	454
0.8	0.8	0.8	0.15	0.3	0.45	373
0.5	0.7	0.9	0.2	0.4	0.6	209
0.5	0.7	0.9	0.9	0.9	0.9	85
0.5	0.7	0.9	0.45	0.3	0.15	335
0.5	0.7	0.9	0.1	0.1	0.1	-

(- işareti eğitim başarılı değil demektir)

İlk 18 denemeden görüldüğü üzere momentum değerleri azdan çoğa doğru gittiği zaman eğitim daha hızlı gerçekleşiyor. Çoktan aza doğru gittiğinde ise eğitim neredeyse gerçekleşmiyor. Hepsinin aynı olduğu durumlarda ise yüksek momentum değerlerinde eğitim gerçekleşiyor ancak düşük momentum değerlerinde eğitim olmuyor.

Eğitim hızında ise 0.15,0.3,0.45 ve 0.45,0.3,0.15 e bakıldığında anlaşılan farklı katmanların eğitim hızlarının farklı olması eğitim için gerekli epok sayısını etkiliyor. Genel olarak yapılabilecek yorum ise eğitim hızı arttığında gerekli epok sayısı azalıyor azaldığında ise gerekli epok sayısı artıyor.

D) Katman sayıları 8,6,4- momentum 0.5,0.7,0.9 – learning rate 0.2,0.4,0.6

Gerçek değerler

1	0	0	0
1	0	0	0
0	0	1	0
0	1	0	0
0	1	0	0
0	0	1	0
0	0	1	0
1	0	0	0
0	0	0	1
0	1	0	0
0	0	0	1
0	1	0	0
0	0	0	1
0	0	1	0
1	0	0	0
0	0	0	1

Tahmin edilen değerler

0.882563	0.0593271	0.075636	1.73857e-06
0.872516	0.0460216	0.0942448	1.31091e-06
0.0836358	0.000150153	0.938781	1.10538e-08
0.0715022	0.864141	0.000220907	0.062464
0.0977823	0.872084	0.000267315	0.0391819
0.0774384	0.000133451	0.944434	9.9898e-09
0.0713574	0.00012063	0.948637	9.30665e-09
0.896187	0.0894293	0.0523008	2.77642e-06
0.00190224	0.127655	4.62808e-05	0.939458
0.0885537	0.870108	0.000251266	0.0455432
0.00180782	0.120491	4.53753e-05	0.943678
0.0430201	0.826591	0.000169736	0.12565
0.00184	0.122912	4.56883e-05	0.942249
0.0720499	0.000121059	0.948569	9.27554e-09
0.902282	0.101986	0.0464774	3.17878e-06
0.00179603	0.119796	4.52278e-05	0.944231

Tahmin değerleri istenen ortalama hata değeri düşürülerek daha iyi hale getirilebilir.

E değerin 0.01 yerine 0.001 seçilirse

0	0	0	1
0	1	0	0
0	1	0	0
0	0	0	1
1	0	0	0
1	0	0	0
0	1	0	0
0	0	0	1
0	0	1	0
0	0	1	0
0	0	0	1
1	0	0	0
0	1	0	0
0	0	1	0
1	0	0	0
0	0	1	0

0.0261641	0.0146733	0.00014476	0.961573
0.000230368	0.97955	2.59386e-05	0.0362332
0.000213949	0.981954	2.52412e-05	0.0327667
0.0336221	0.0100137	0.000163791	0.965334
0.965258	1.04554e-06	0.0235741	0.0288955
0.967022	7.82921e-07	0.0304128	0.020754
0.000214292	0.98189	2.52537e-05	0.032914
0.0257539	0.0151164	0.000144369	0.959989
0.0234821	2.32761e-08	0.974874	3.97788e-05
0.0272767	2.57152e-08	0.970177	4.66405e-05
0.0402466	0.00758621	0.000177661	0.968796
0.96299	1.01429e-06	0.0252678	0.0267795
0.000228207	0.979864	2.58368e-05	0.0358649
0.0237769	2.37823e-08	0.974234	4.06538e-05
0.96462	8.29178e-07	0.0299592	0.0214965
0.0255319	2.54305e-08	0.971531	4.47984e-05

Soru 1.2

Veriler iris.data dosyasından okunduktan sonra eğitim ağına sunuldu. Yani soru 1.1'deki kurulmuş olan ağda ufak değişikler yapılarak çözüldü.

Iris Setosa'ya 100, Iris Versicolor'a 010, Iris Virginica'ya 001 Hamming değerleri verilerek ağa sunuldu.

Katman Nöronları: 50,30,3; Momentum: 0.5,0.7,0.9; Learning rate: 0.1,0.2,0.3; Ortalama hata: 0.001 olarak seçildi. Veriler bu sefer gerçek değerler olduğu için az katman sayısı eğitmeye yeterli olmadı. Bu yüzden katman sayısı arttırıldı. Daha karmaşık bir veri kümesi olduğu için yüksek learning rate kullanarak eğitim sonuçları kötü geldiği için düşük learning rate büyüklükleri seçildi.Daha iyi bir eğitim elde edebilmek adına ortalama hata değeri daha düşük seçildi. Son olarak ise eğitim ve test kümeleri belirlendi. Toplamda elimizde olan 150 veri 120 eğitim ve 30 test kümesi olarak 2 ye ayrıldı.

Gerçek ve Tahmin değerleri

0	1	0
1	0	0
1	0	0
1	0	0
1	0	0
0	0	1
1	0	0
0	1	0
0	1	0
1	0	0
0	0	1
0	0	1
1	0	0
0	1	0
0	0	1

0	3.815e-05	0.546727	0.087852
1	1	0.000412635	9.83815e-28
2	0.999982	0.0282201	4.88852e-23
3	0.999999	0.00236589	8.74033e-26
4	0.999999	0.00121366	1.56337e-26
5	2.03165e-06	0.0238169	0.992275
6	0.999993	0.0121924	5.90335e-24
7	1.16238e-05	0.220305	0.637355
8	0.00041234	0.927644	0.000299705
9	0.999999	0.0027675	1.24667e-25
10	2.20638e-06	0.0271323	0.990536
11	1.48416e-06	0.0154983	0.996417
12	0.999998	0.00456914	4.50209e-25
13	3.37966e-05	0.504915	0.11487
14	9.67586e-08	0.000271973	0.999996

0	1	0
0	1	0
0	0	1
0	1	0
0	1	0
0	0	1
0	0	1
0	0	1
0	0	1
0	0	1
0	1	0
0	1	0
1	0	0
0	0	1
1	0	0

15	1.90186e-05	0.334929	0.346037
16	0.000279231	0.901253	0.000768282
17	4.03332e-07	0.00222529	0.999856
18	4.73176e-06	0.0774604	0.941061
19	0.000154735	0.839864	0.0032
20	6.37666e-07	0.00436039	0.999555
21	1.04671e-06	0.00884812	0.998491
22	3.63847e-07	0.00186566	0.999889
23	3.91256e-07	0.00209442	0.999867
24	2.57351e-07	0.00113466	0.999953
25	0.000191819	0.863825	0.00190724
26	0.000507441	0.935043	0.00018274
27	1	0.000445685	1.17256e-27
28	2.97524e-07	0.0013973	0.999932
29	1	7.84719e-05	1.3127e-29

(Soldaki değerler gerçek değerler, Sağda bulunanlar ise tahmin değerleri)

Soru 2

Sistem tanıma problemini çözebilmek için oluşturduğumuz ağı Billings sisteminin çıkışları ile eğitmemiz gerekir. Çıkış verilerimizin Billings sisteminin çıkışlarından farkını alarak ağırlıkları güncellememiz gerekir. Negatif ve pozitif değerler alabilen Billings sistem çıkışları olduğundan dolayı aktivasyon fonksiyonunun değiştirilmesi gerekir.

3.SORU

*Bu sorunun çözümünde tensorflow kütüphanesi kullanılmıştır.

İris.data verileri pandas kütüphanesinde bulunan pd.read_csv komutu yardımıyla sisteme aktarıldı. Sonrasında df.sample komutu verilerimizin karıştırılması için kullanıldı. Bu sayede her denemede farklı bir veri seti üzerinde test etme imkanı oluştu. Değişken verileri X_Data, tür verileri Y_Data altında toplandı. Değişken verilerinin 0 ve 1 arasında değerler almaları için veriler içlerindeki maksimum değerlere bölünerek normalizasyon işlemi yapıldı. Tür verileri sırasıyla 0, 1 ve 2 integer değerleri ile eşleştirildi. Train_test_split komutu yardımıyla verilerin % 75'i eğitilip kalan % 25'lik kısımda test edildi. Her değişken için bir feature column oluşturuldu ve tek bir array içerisinde toplandı. Sonrasında train_func, test_func, predict_func ve model oluşturulup eğitildi.

133	0.797468	0.636364	0.73913	0.6	133	2
115	0.810127	0.727273	0.768116	0.92	115	2
102	0.898734	0.681818	0.855072	0.84	102	2
112	0.860759	0.681818	0.797101	0.84	112	2
60	0.632911	0.454545	0.507246	0.4	60	1
55	0.721519	0.636364	0.652174	0.52	55	1
127	0.772152	0.681818	0.710145	0.72	127	2
39	0.64557	0.772727	0.217391	0.08	39	0
18	0.721519	0.863636	0.246377	0.12	18	0
100	0.797468	0.75	0.869565	1	100	2
19	0.64557	0.863636	0.217391	0.12	19	0
126	0.78481	0.636364	0.695652	0.72	126	2
147	0.822785	0.681818	0.753623	0.8	147	2
146	0.797468	0.568182	0.724638	0.76	146	2
132	0.810127	0.636364	0.811594	0.88	132	2
113	0.721519	0.568182	0.724638	0.8	113	2
79	0.721519	0.590909	0.507246	0.4	79	1
68	0.78481	0.5	0.652174	0.6	68	1
8	0.556962	0.659091	0.202899	0.08	8	0
33	0.696203	0.954545	0.202899	0.08	33	0
28	0.658228	0.772727	0.202899	0.08	28	0
88	0.708861	0.681818	0.594203	0.52	88	1
16	0.683544	0.886364	0.188406	0.16	16	0
11	0.607595	0.772727	0.231884	0.08	11	0
10	0.683544	0.840909	0.217391	0.08	10	0
5	0.683544	0.886364	0.246377	0.16	5	0
136	0.797468	0.772727	0.811594	0.96	136	2
7	0.632911	0.772727	0.217391	0.08	7	0
54	0.822785	0.636364	0.666667	0.6	54	1
37	0.620253	0.704545	0.217391	0.04	37	0

(x_test, y_test)

133	0.797468	0.636364	0.73913	0.6	133	2
115	0.810127	0.727273	0.768116	0.92	115	2
102	0.898734	0.681818	0.855072	0.84	102	2

(X_Data,Y_Data)

Gerçek değerler test sonucu değerleri ile karşılaştırıldığında çok küçük istisnalar dışında aynı sonucu vermektedir.

Sonuçta ağımızın 0.966 doğruluk oranıyla çalıştığını görmekteyiz.

accuracy	float32	1	0.96666664
average_loss	float32	1	0.22654437
global_step	int64	1	1000
loss	float32	1	6.796331

(Results)

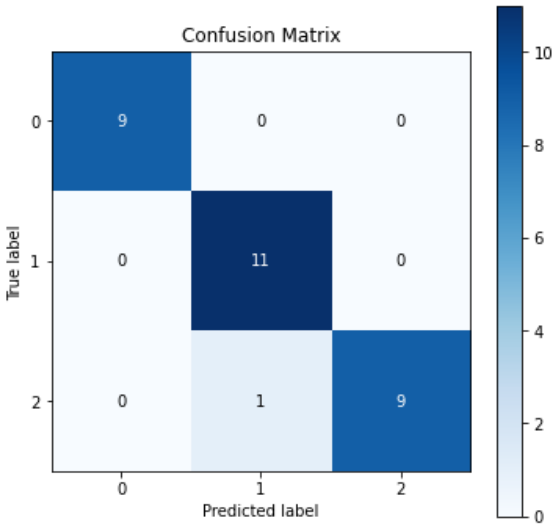
	precision	recall	f1-score	support
0	1.00	1.00	1.00	12
1	0.86	1.00	0.92	6
2	1.00	0.92	0.96	12
accuracy			0.97	30
macro avg	0.95	0.97	0.96	30
weighted avg	0.97	0.97	0.97	30

(Prediction_report)

Index	SepalLenqthCm	SepalWidthCm	PetalLenqthCm	PetalWidthCm	Index	Species
107	0.924051	0.659091	0.913043	0.72	107	2
142	0.734177	0.613636	0.73913	0.76	142	2
81	0.696203	0.545455	0.536232	0.4	81	1
93	0.632911	0.522727	0.478261	0.4	93	1
51	0.810127	0.727273	0.652174	0.6	51	1
143	0.860759	0.727273	0.855072	0.92	143	2
117	0.974684	0.863636	0.971014	0.88	117	2
26	0.632911	0.772727	0.231884	0.16	26	0
67	0.734177	0.613636	0.594203	0.4	67	1
61	0.746835	0.681818	0.608696	0.6	61	1
105	0.962025	0.681818	0.956522	0.84	105	2
139	0.873418	0.704545	0.782609	0.84	139	2
86	0.848101	0.704545	0.681159	0.6	86	1
90	0.696203	0.590909	0.637681	0.48	90	1
13	0.544304	0.681818	0.15942	0.04	13	0

(x_train, y_train)

Daha sonra prediction ve y_test verilerinden yararlanılarak Confusion Matrix oluşturuldu.



Index	Species
107	2
0	0
66	1
25	0
86	1
103	2
88	1
51	1
114	2
112	2
17	0
1	0
94	1
135	2

Indi	Type	Size	
0	int64	1	2
1	int64	1	0
2	int64	1	1
3	int64	1	0
4	int64	1	1
5	int64	1	2
6	int64	1	1
7	int64	1	1
8	int64	1	2
9	int64	1	2
10	int64	1	0
11	int64	1	0
12	int64	1	1
13	int64	1	2

(y_test, prediction)

Elde ettiğimiz veriler karşılaştırıldığında Tensorflow kütüphanesi yardımıyla kurduğumuz ağın hata payı kendi oluşturduğumuz ağın hata payına oldukça yakın çıkmaktadır. Bu sayede ağımızın doğru çalıştığını anlamaktayız.