GEREKSİNİM RAPORU

Giriş:

Günümüzde internet ağları ve sosyal medya kullanımı, insanların başka kişi ya da kurumlar hakkındaki görüşlerini kolay ve açık şekilde ifade edebilmesine ve yorumlamasına imkân sağlamıştır. Bu imkânın giderek yaygınlaşması ile büyüyen zengin bir içerik ortaya çıkmaktadır. Bunun sonucunda internetten elde edilen büyük verinin analiz edilerek anlamlı bilgiye dönüştürülmesi ve kullanılması son yıllarda yoğun olarak çalışılan bir konudur. Bu bağlamda, son on yılda, sosyal medya içeriğini analiz etmek için yapılan çalışmalar hızlı bir artış göstermiş ve netnografi çalışmalarından (Kozinets, 2009; Hine, 2005) fikir madenciliğine (Dave vd., 2003) ve doğal dil işlemeye (Nasukawa ve Yi, 2003) kadar çok sayıda yöntem önerilmiştir. Duygu analizi ise büyük veri içindeki öznel bilginin sistematik olarak analiz edilmesini sağlayan bir doğal dil işleme yöntemidir. Metin madenciliği ve hesaplamalı dilbilim araçlarını kullanarak, araştırmacıların metin içindeki duyguları belirlemelerine, duygu kategorilerine göre sınıflandırmalarına ve kategorilerin duygusal polaritelerini bulmalarına yardımcı olur (Arazy ve Woo, 2007; Jacobson, 2009).

Python ile Film Yorumlarının Duygu Analizi

Duygu Analizi asıl olarak, insani duyguları makineye öğretmeyi amaçlamaktadır. Çoğunlukla pozitif, negatif ve nötr olarak sınıflandırma yapılır. Fakat başka birçok çeşitleri de mevcuttur. Mesela mutlu, üzgün, şaşkın, kızgın vb. şeklinde de sınıflandırma yapılabilir.

Son zamanlarda çok fazla önem kazanan bir alan oldu. Özellikle pandemi süreci içerisinde yorumların önemi kat kat arttı. İnsanlar sosyal medya, film/dizi platformları, e-ticaret vb. sitelerde oldukça fazla vakit geçirmeye başladı. Bunlardan birisi için duygu analizi örneği verecek olursak; e-ticaret sitelerinde herhangi bir ürünün alışverişinden sonra yapılan yorumlar alıcının o ürün hakkındaki duygularını, tutumunu belirlemeye fayda sağlar. Bu sayede satıcı müşteri tutumunu tespit edip yapılması gereken herhangi bir şey varsa yapmak ya da düzeltilmesi gereken herhangi bir durum varsa düzeltmek konusunda girişimlerde bulunur. Aynı zamanda bahsettiğimiz üzere özellikle

pandemi sürecinde diğer alıcı adaylara büyük fayda sağlamıştır. Çünkü müşteriler ürünü canlı göremiyorlar ve bu yüzden de tereddüt içerisinde kalıyorlar. Mesela kıyafet alırken tam olarak beden yapısı hakkında öngörüşlü olamıyorlar ya da ev eşyası alacaklarsa kalitesi hakkında bir çıkarımda bulunamıyorlar. Bu yorumlar sayesinde ürünün kalitesi, tipi, yapısı, faydası, fiyatı vb. özellikleri hakkında fikir edinip ürünü alıp almamaları konusunda daha kolay karara varabiliyorlar. Bu da yine satıcının satış miktarını büyük oranda etkilemektedir.

Film Yorumlarının Duygu Analizi

Kaggle üzerinden aldığım veri seti içerisinde 83227 gözlem ve 3 değişken mevcuttur. Bu veri; film yorumları (comments), film adları (film_name) ve yorumların duygu derecelerinden (points) oluşmaktadır.

Öncelikle veriyi ön işlemlerden geçirip modele uygun hale getireceğiz ve ardından duygu analizini gerçekleştireceğiz.

Gerekli kütüphaneleri import edelim.

```
import re
import string
import numpy as np
import pandas as pd

from tensorflow.keras.optimizers import Adam
from tensorflow.python.keras.models import Sequential
from tensorflow.python.keras.preprocessing.text import Tokenizer
from tensorflow.python.keras.layers import Embedding, GRU, Dense
from tensorflow.python.keras.preprocessing.sequence import pad_sequences
```

Veri setimizi okutalım.

```
df = pd.read_csv("turkish_movie_sentiment_dataset.csv")
df.head()
```

	comment	film_name	point
0	\n Jean Reno denince zate...	Sevginin Gücü	5,0
1	\n Ekşın falan izlemek is...	Sevginin Gücü	5,0
2	\n Bu yapım hakkında öyle...	Sevginin Gücü	5,0
3	\n finali yeter... (sting...	Sevginin Gücü	5,0
4	\n Jean Reno..\r\nbu adam...	Sevginin Gücü	5,0

Görüldüğü üzere yorumların hepsinin başında “\n” böyle bir karakter mevcuttur. Biz de bir fonksiyon yazarak bu karakterleri kaldıracağız.

Birkaç satır ile kontrol ettiğimde verinin başından 23 karakter ileri, sonundan da 24 karakter geri gittiğimde oluşmuş boşluklar ve “\n” karakterleri ortadan kalkıyor. O yüzden bunu tüm veriye uygulayabileceğimiz bir fonksiyon yazmamız gerekiyor.

Öncelikle comments adlı bir fonksiyon oluşturuyorum. Bu fonksiyon istediğimiz değişken içerisindeki gözlemlerin başından 23, sonundan 24 karakter kaldıracak. Ardından bu fonksiyonu comment değişkenindeki gözlemlere apply modülü ile tek tek uyguluyorum ve tekrar aynı değişken içerisine atıyorum.

```
comments = lambda x : x[23:-24]

df["comment"] = df["comment"].apply(comments)
df["comment"].head()
```

```
0    Jean Reno denince zaten leon filmi gelir akla ...
1    Ekşın falan izlemek istiyorsanız eğer bunu izl...
2    Bu yapım hakkında öyle çok şey yazabilirim ki ...
3    finali yeter... (sting - shape of my heart)\r\...
4    Jean Reno..\r\nbu adam kusursuz biri..\r\nve o...
Name: comment, dtype: object
```

```
In [16]: df["point"].unique()

Out[16]: array(['5,0', '3,5', '4,5', '2,5', '2,0', '3,0', '4,0', '1,5', '0,5',
                '1,0', '4,6', '3,9', '3,8', '3,7', '3,2', '3,1'], dtype=object)
```

Filmlere 1–5 arası puanlandırma yapılmıştır. Bizim duygu sınıflandırmamız pozitif ve negatif olarak ayrılacağı için bunu ikili sınıflandırmaya dönüştürmem gerekiyor. Fakat ondan önce bir sorun daha mevcut, veri içerisinde numaralar virgül ile ayrılmış. Biz sadece ilk karakteri alacağız ve ardından float tipine dönüştüreceğiz.

Bir önceki kodda yaptığımıza benzer floatize fonksiyonu oluşturuyoruz ve point içerisindeki gözlemleri apply modülünü kullanarak düzeltiyoruz.

```
floatize = lambda x : float(x[0:-2])

df["point"] = df["point"].apply(floatize)
df["point"].value_counts()
```

```
4.0    27463
5.0    15873
3.0    14494
2.0    13866
1.0     6381
0.0     5150
Name: point, dtype: int64
```

Düzelttiğimiz puanları, 0 ve 1 olarak sınıflandıracağız. 3 puanı bizim için nötr oluyor. Ne pozitif ne de negatif diyebiliyoruz. Bu satırları veriden çıkaracağız. Ardından 3'ten küçüklere 0 (negatif), 3'ten büyüklere 1 (pozitif) diye atama yapacağız.

```
df.drop(df[df["point"] == 3].index, inplace = True)
df["point"] = df["point"].replace(1, 0)
df["point"] = df["point"].replace(2, 0)
df["point"] = df["point"].replace(4, 1)
df["point"] = df["point"].replace(5, 1)
df["point"].value_counts()
```

```
1.0    43336
0.0    25397
Name: point, dtype: int64
```

Veri içerisinden sildiğimiz satırlar olduğu için index yapısı bozuldu. İndexleri resetliyoruz.

```
df.reset_index(inplace = True)
df.drop("index", axis = 1, inplace = True)
df.head()
```

	comment	film_name	point
0	Jean Reno denince zaten leon filmi gelir akla ...	Sevginin Gücü	1.0
1	Ekşın falan izlemek istiyorsanız eğer bunu izl...	Sevginin Gücü	1.0
2	Bu yapım hakkında öyle çok şey yazabilirim ki ...	Sevginin Gücü	1.0
3	finali yeter... (sting - shape of my heart)\r\...	Sevginin Gücü	1.0
4	Jean Reno..\r\nbu adam kusursuz biri..\r\nve o...	Sevginin Gücü	1.0

Yorumların ön işlemlerine geçiş yapıyoruz. İlk önce comments altındaki bütün karakterleri küçültüyoruz. Bunu lower modülü ile gerçekleştireceğiz. Apply modülü ile tüm veri içerisine uyguluyoruz.

```
df["comment"] = df["comment"].apply(lambda x: x.lower())
df.head()
```

	comment	film_name	point
0	jean reno denince zaten leon filmi gelir akla ...	Sevginin Gücü	1.0
1	ekşın falan izlemek istiyorsanız eğer bunu izl...	Sevginin Gücü	1.0
2	bu yapım hakkında öyle çok şey yazabilirim ki ...	Sevginin Gücü	1.0
3	finali yeter... (sting - shape of my heart)\r\...	Sevginin Gücü	1.0
4	jean reno..\r\nbu adam kusursuz biri..\r\nve o...	Sevginin Gücü	1.0

Şimdi ise noktalama işaretlerini kaldıracağız. remove_punctuation adlı bir fonksiyon yazıyoruz. Bu fonksiyon yorumlar içerisinde dolanacak ve eğer gezindiği karakter noktalama işareti değil ise bunu word_wo_punc değişkenine atacak. Ardından da biz bu apply modülü ile bunu veriye uygulayacağız ve bize veriyi noktalama işaretleri kaldırılmış bir şekilde verecek. Fakat bunu tek başına yaptığımızda içerisinde hala kalkmamış olan “\n” ve “\r” karakterleri yer alıyor. Bunu da kendimiz manuel olarak değiştiriyoruz.

```
def remove_punctuation(text):
    no_punc = [words for words in text if words not in string.punctuation]
    word_wo_punc = "".join(no_punc)
    return word_wo_punc

df["comment"] = df["comment"].apply(lambda x: remove_punctuation(x))
df["comment"] = df["comment"].apply(lambda x: x.replace("\r", " "))
df["comment"] = df["comment"].apply(lambda x: x.replace("\n", " "))

df.head()
```

	comment	film_name	point
0	jean reno denince zaten leon filmi gelir akla ...	Sevginin Gücü	1.0
1	ekşın falan izlemek istiyorsanız eğer bunu izl...	Sevginin Gücü	1.0
2	bu yapım hakkında öyle çok şey yazabilirim ki ...	Sevginin Gücü	1.0
3	finali yeter sting shape of my heart bazıl...	Sevginin Gücü	1.0
4	jean reno bu adam kusursuz biri ve oyunculug...	Sevginin Gücü	1.0

Sırada numerik karakterleri kaldırmamız gerekiyor. Bunun içinde remove_numeric isimli bir fonksiyon yazıyoruz. Bu fonksiyon da yorumlar içerisinde dolanacak ve numerik olmayan her şeyi output değişkeninin içerisine atacak. Ardından bize de numerik karakterler kaldırılmış olarak bir çıktı verecek.

```
def remove_numeric(corpus):
    output = "".join(words for words in corpus if not words.isdigit())
    return output

df["comment"] = df["comment"].apply(lambda x: remove_numeric(x))
df.head()
```

	comment	film_name	point
0	jean reno denince zaten leon filmi gelir akla ...	Sevginin Gücü	1.0
1	ekşın falan izlemek istiyorsanız eğer bunu izl...	Sevginin Gücü	1.0
2	bu yapım hakkında öyle çok şey yazabilirim ki ...	Sevginin Gücü	1.0
3	finali yeter sting shape of my heart bazıl...	Sevginin Gücü	1.0
4	jean reno bu adam kusursuz biri ve oyunculug...	Sevginin Gücü	1.0

Yorumlar Word Cloud

Artık modeli kurmak için girişimlerde bulunabiliriz. Etiketleri (points) ve yorumları (comments) ayırarak listeler içerisine alıyoruz.

Elimizdeki verilerin %80'i ile modeli kurup, %20'si ile modelimizi test edeceğiz. Bu sebepten dolayı öncelikle verinin boyutunun %80'inini alarak cutoff değişkenine atıyoruz. Ardından da cutoff değişkenine göre oluşturduğumuz target ve data listelerini, train ve test olarak ayırıyoruz.

```
target = df["point"].values.tolist()
data = df["comment"].values.tolist()

cutoff = int(len(data)*0.80)

X_train, X_test = data[:cutoff], data[cutoff:]
y_train, y_test = target[:cutoff], target[cutoff:]
```

Artık elimizdekileri tokenleştirebiliriz. Tokenleştirme işlemi yapılırken her yorum kelimelere ayrılacak. Kelime hazinesindeki her kelimeye karşılık farklı bir sayı gelecek. İlk önce kelime haznemizde en fazla kaç tane kelime oluşsun bunu belirleyeceğiz. Bunun için num_words değişkenine 10.000 değerini veriyoruz. Bununla beraber en sık geçen 10.000 kelimeyi alacağız. Duygu Analizi yapacağımız için veri içerisindeki tüm kelimeleri almamıza gerek yok. Tokenleştirme işlemini “keras” kullanarak gerçekleştiriyoruz.

```
num_words = 10000
tokenizer = Tokenizer(num_words = num_words)
tokenizer.fit_on_texts(data)
# tokenizer.word_index
```

Tokenizer tanımladığımıza göre elimizdeki veriyi tokenleştirebiliriz.

Train ve test verisindeki tüm yorumları tokenler halinde bir değişken içerisinde saklayalım. Ardından da 1000. satırın nasıl gözüktüğünü kontrol edelim.

```
X_train_tokens = tokenizer.texts_to_sequences(X_train)
X_test_tokens = tokenizer.texts_to_sequences(X_test)

print([X_train[1000]])
print(X_train_tokens[1000])
```

```
['film gerçekten bomba   çok komikti eğlenceli vakit geçirmek isteyen herkese öneririm  eddie murphy de gerçekten super oynamı']
[3, 35, 3225, 5, 1857, 128, 202, 672, 872, 269, 1641, 3232, 3495, 7, 35, 1327, 8852]
```

Kelime haznesini sınırlandırdığımız için her kelimeye karşılık bir token bulunmayacak. Eğer kelime haznesinde değilse o kelime, yok sayılayacak.

Sinir ağlarında genellikle RNN ile oluşturduğumuz modellere belli boyutlarda inputlar veririz. Bizim verimizdeki yorumların hepsi farklı sayılardaki kelimelerden oluşmaktadır. Farklı sayılardaki kelimeleri RNN içerisine input olarak veremeyiz. Tüm yorumları aynı boyuta getirmemiz gerekir. Eğer yorum belirlediğimiz boyuttan düşük boyutta olursa 0 eklenecek. Eğer yorum belirlediğimiz boyuttan yüksekse belirli kısımları silerek kendi belirlediğimiz boyuta getireceğiz.

Öncelikle for döngüsü oluşturup veri setimizdeki her yorumun üzerinden tek tek geçeceğiz ve token sayısını alacağız. Liste üzerinde işlem yapmayı kolaylaştırmak amacıyla da listeyi numpy array'e dönüştüreceğiz.

```
num_tokens = [len(tokens) for tokens in X_train_tokens + X_test_tokens]
num_tokens = np.array(num_tokens)
num_tokens
```

```
array([14, 47, 96, ..., 21,  4, 51])
```

Ortalama olarak bir yorumda kaç tane token olduğuna bakalım.

```
np.mean(num_tokens)
```

```
31.318813379308338
```

En fazla token bulunan yorumun token sayısına bakalım.

```
np.max(num_tokens)
```

```
3094
```

Şimdi yorumların boyutlarını eşitleyeceğiz. Bunun için kendimiz bir boyut belirliyoruz. Kendimiz değer verebiliriz, en yüksek tokenli yorumun boyutunu kullanabiliriz ya da ortalamayı alabiliriz. Biz ise ortalamayı alıp 2 standart sapma ekleyerek elde edeceğiz.

```
max_tokens = np.mean(num_tokens) + (2*np.std(num_tokens))
max_tokens = int(max_tokens)
max_tokens
```

```
141
```

Elde ettiğimiz boyut yorumların yüzde kaçını kapsıyor bakalım.

```
np.sum(num_tokens < max_tokens) / len(num_tokens)
```

```
0.9741754324705746
```

Train setindeki her yorumun boyutunu elde ettiğimiz boyuta getiriyoruz. İlk önce padding eklenecek veriyi giriyoruz. Ardından belirlediğimiz boyutu giriyoruz.

```
X_train_pad = pad_sequences(X_train_tokens, maxlen = max_tokens)
X_test_pad = pad_sequences(X_test_tokens, maxlen = max_tokens)

print(X_train_pad.shape)
print(X_test_pad.shape)
```

```
(54986, 141)
(13747, 141)
```

Padding'i daha iyi görebilmek 800. yoruma bakalım.

```
np.array(X_train_tokens[800])
```

```
array([  41,   36, 1864,  416,   54, 1824,  480,  355,   80,   33])
```

Aynı yorumun padding uygulanmış haline bakalım.

```
X_train_pad[2000]
```

```
array([   0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
          0,    0,    0,    0,    0,    0,    5,   70,    1,    3,    4,
       5061, 4071,  851,    5,    5,   16,  527, 2370, 1105])
```

Tokenleri verip, stringi alabilmek için fonksiyon yazacağız. Bu şekilde elimizdeki tokenleri tekrar yoruma çevireceğiz. word_index içerisinde sözlük olarak kelimeler, kelimelerin sayısal olarak tokenleri bulunuyordu. word_index ile kelimeler ve sayıların yerini tersine çeviriyoruz. Artık bir sayıyı verdiğimiz zaman o sayıya karşılık gelen kelimeyi bulabileceğiz. Fonksiyon içerisinde sıfırlar kelimeye karşılık gelmediği için dahil etmiyoruz. Diğerlerini de liste içerisinde topluyoruz. Son olarak da bize yorumu çıktı olarak verecek.

```
idx = tokenizer.word_index
inverse_map = dict(zip(idx.values(), idx.keys()))

def tokens_to_string(tokens):
    words = [inverse_map[token] for token in tokens if token != 0]
    text = " ".join(words) # Kelimeler aralarında boşluk bırakılarak ard arda
    return text
```

```
tokens_to_string(X_train_tokens[350])
```

```
'bence gayet güzel bi filmdi her ne kadar kendi dışına yinede bence mükemmel
bi oyunculuk ama her seyrettiğimde ayrı güzel bu filmide korku kategorisine
hayret ettim doğrusu ne'
```

Sıra modeli kurmaya geldi. Embedding matrisini oluşturmadan önce, embedding matrisinin büyüklüğünü belirliyoruz. Her kelimeye karşılık gelen 50 uzunluğunda vektör oluşturuyoruz. Modele bir şey ekleyebilmek model.add kullanıyoruz.

num_words (10 bin) kelime sayılarıdır. 10 bin kelimeye karşılık gelen 50 uzunluğunda rastgele vektörler oluşturalım. Layer input aldığı zaman, input içerisindeki kelimelerin vektörlerini bir sonraki layer'a gönderecektir. embedding_layer'ın output'u, bir sonraki layer'ın input'u olacak.

Yinelenen sinir ağımızı (RNN) oluşturuyoruz. Yinelenen sinir ağını oluşturmak GRU kullanacağız. model.add(GRU()) bu şekilde de kullanılabilir. units, GRU değerimizin nöron sayısıdır. Bu layer'da 16 tane output verilecek. return_sequence = True, output olarak sequences tamamı döndürülecek. Eğer False olsaydı sadece son output döndürülürdü. Bir sonraki layer'a GRU ekleyeceğimiz için buna True demek zorundayız. Eğer bir sonraki layer bir nörondan oluşsaydı False girerdik. Sigmoid ile output'u [0,1] arasına sıkıştırıyoruz.

```
embedding_size = 50
model = Sequential()
model.add(Embedding(input_dim = num_words, output_dim = embedding_size, input
model.add(GRU(units = 16, return_sequences = True))
model.add(GRU(units = 8, return_sequences = True))
model.add(GRU(units = 4))
model.add(Dense(1, activation = "sigmoid"))
```

Modeli kurduk. Modelin eğitimine dair optimizasyon algoritması belirleyeceğiz. Bunun için Adam algoritmasını kullanacağız.

Optimizasyon algoritmasını belirledikten sonra, modelimizi derliyoruz. Sadece iki sınıf olduğu için de "binary_crossentropy" kullanacağız.

```
embedding_size = 50
model = Sequential()
model.add(Embedding(input_dim = num_words, output_dim = embedding_size, input_length = max_tokens, name = "embedding_layer"))
```

```
optimizer = Adam(lr = 1e-3)
model.compile(loss = "binary_crossentropy", optimizer = optimizer, metrics = ["accuracy"])
model.summary()
```

```
optimizer = Adam(lr = 1e-3)
model.compile(loss = "binary_crossentropy", optimizer = optimizer, metrics =
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_layer (Embedding)  (None, 141, 50)           500000
_________________________________________________________________
gru (GRU)                    (None, 141, 16)           3216
_________________________________________________________________
gru_1 (GRU)                  (None, 141, 8)            600
_________________________________________________________________
gru_2 (GRU)                  (None, 4)                 156
_________________________________________________________________
dense (Dense)                (None, 1)                 5
=================================================================
Total params: 503,977
Trainable params: 503,977
Non-trainable params: 0
_________________________________________________________________
```

Eğitime başlıyoruz. Eğitimden kaç defa geçeceğine epochs parametresi ile karar veriyoruz.

```
X_train_pad = np.array(X_train_pad)
y_train = np.array(y_train)

model.fit(X_train_pad, y_train, epochs = 5, batch_size = 256)
```

```
Epoch 1/5
215/215 [==============================] - 46s 213ms/step - loss: 0.5216 -
accuracy: 0.7567
Epoch 2/5
215/215 [==============================] - 45s 208ms/step - loss: 0.4076 -
accuracy: 0.8276
Epoch 3/5
215/215 [==============================] - 44s 207ms/step - loss: 0.3704 -
accuracy: 0.8509
Epoch 4/5
215/215 [==============================] - 44s 204ms/step - loss: 0.3526 -
accuracy: 0.8625
Epoch 5/5
215/215 [==============================] - 44s 205ms/step - loss: 0.3385 -
accuracy: 0.8715
<tensorflow.python.keras.callbacks.History at 0x1da83f5f550>
```

Test setindeki ilk 1000 yorum için tahminde bulunuyoruz. Bu tahminler bir matrisin sütunu olarak atanacaktır. Her satırda bir yorumun tahmin değeri bulunacaktır. Bu elemanlarla işlem yapmayı kolaylaştırabilmek için sütunu satıra çevireceğiz. Böylece elimizde tahmin değerlerini gösterecek bir vektör elde edeceğiz. Vektör üzerinde işlem yapmak daha kolaydır. Sütunu satıra çevirebilmemiz için matrisin transpozunu alıyoruz.

```
y_pred = model.predict(X_test_pad[0:1000])
y_pred = y_pred.T[0]
# y_pred
```

Elimizdeki tahmin değerlerini gerçekleri ile karşılaştırarak hangi yorumlarda yanlış olduğunu göreceğiz. Elimizdeki tahmin değerleri 0.5'ten büyükse 1, küçükse 0 yapacağız. Böylece 0 ve 1'lerden oluşan bir vektör elde edeceğiz. Elde ettiğimiz vektörü etiketlerle karşılaştıracağız.

```
cls_pred = np.array([1.0 if p > 0.5 else 0.0 for p in y_pred])
cls_true = np.array(y_test[0:1000])
```

Elimizdeki iki vektörü karşılaştıracağız. Yanlış tahminlerin indexlerini alalım.

```
incorrect = np.where(cls_pred != cls_true)
incorrect = incorrect[0]
# incorrect
```

1000 tane yorumdan kaç tanesinin yanlış tahmin edildiğine bakalım.

```
len(incorrect)
```

```
277
```

Yanlış bilinen yorumlardan bir yorumun indexini alarak yoruma bakalım.

```
idx = incorrect[10]
X_test[idx]
```

```
'robert ludlumun romanları gerçekten sürükleyiciydibourne serisi sinamada
bunu kanıtladıyazarın ikinci dünya savaşı ve özelliklede nazileri konu alan
kitaplarıda çok güzeldirbu filmi henüz izleme fırsatımız olmadı ama tom
cruisedan nasıl bir rus ajanı olur o tartışılı'
```

Modelimizin bu yoruma nasıl tahminde bulunduğuna bakalım.

```
y_pred[idx]
```

```
0.87903625
```

Modelimiz yorumu olumlu tahmin etmiş, doğrusuna bakalım.

```
cls_true[idx]
```

```
0.0
```

Kütüphaneler:

libraries & frameworks

Install NumPy with pip:

```
pip install numpy
```

Install pandas with pip:

```
pip install pandas
```

Install scikit-learn with pip:

```
pip install scikit-learn
```

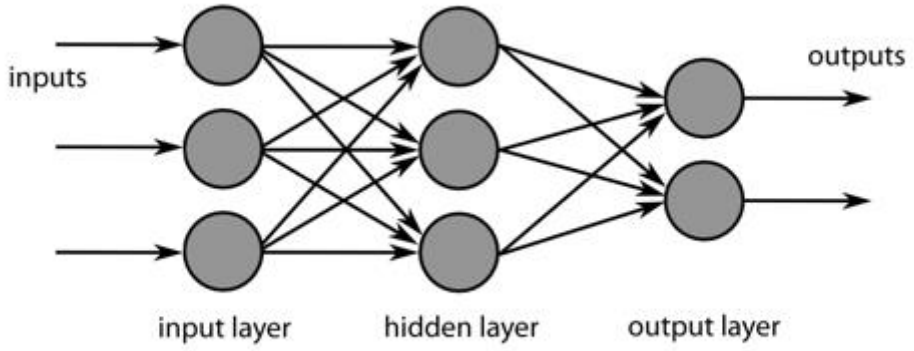
Install matplotlib with pip:

```
pip install matplotlib
```

Recurrent Neural Network Nedir?

Recurrent Neural Networks (RNN) ‘ü anlayabilmek için önce feedforward (ileri doğru çalışan) bir ağın çalışma prensibini tekrar inceleyelim. Kısaca katmanlar üzerindeki nöronlara gelen bilgilere bir takım matematiksel işlemler uygulayarak çıktı üreten bir yapı diyebiliriz.

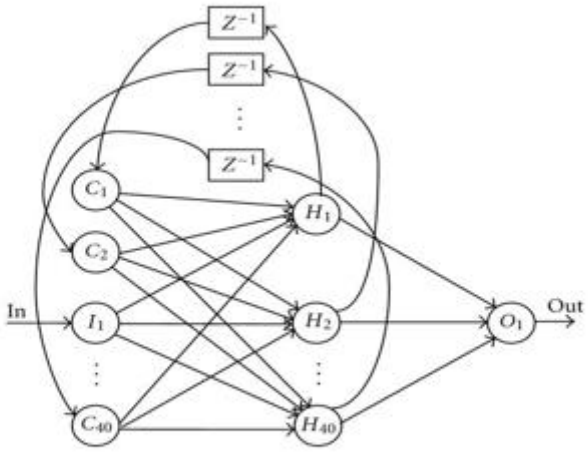
Feedforward çalışan yapıda gelen bilgi sadece ileri doğru işlenir. Bu yapıda kabaca, input verileri ağdan geçirilerek bir output değeri elde edilir. Elde edilen output değeri doğru değerler ile karşılaştırılarak hata elde edilir. Ağ üzerindeki ağırlık değerleri hataya bağlı olarak değiştirilir ve bu şekilde en doğru sonucu çıktı verebilen bir model oluşturulmuş olur.



FeedForward Neural Network Yapısı

Feedforward bir ağın eğitiminde hatanın yeterince düşürülmesi gerekir. Böylece nöronlara giden ağırlıklar yenilenerek girilen inputa uygun output verecek bir yapı oluşmuş olur.

Örneğin, bir fotoğraf üzerindeki nesneleri kategorize etmek için eğitilen feedforward bir ağ düşünelim. Verilen fotoğraf rastgele bir sıra ile de olsa hem eğitim hem de test için kullanılabilir. Bir önceki veya bir sonraki fotoğraf ile herhangi bir bağının olması gerekmez. Yani zamana veya sıraya bağlı bir kavram yoktur, ilgilendiği tek input o andaki mevcut örnektir.

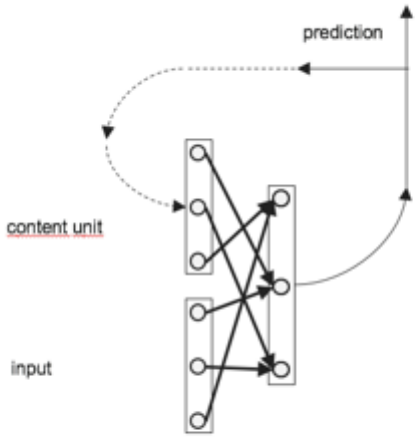


Recurrent Neural Network Yapısı

Recurrent(yinelenen) yapılarda ise sonuç, sadece o andaki inputa değil, diğer inputlara da bağlı olarak çıkarılır. Şekilde de görüleceği gibi RNN'de t anındaki input verilerinin yanında, t-1 anından gelen hidden layer sonuçları da hidden layer'ın t anındaki girdisidir. t-1 anındaki input için verilen karar, t anında verilecek olan kararı da etkilemektedir. Yani bu ağlarda inputlar şimdiki ve önceki bilgilerin birleştirilmesi ile output üretirler.

Recurrent yapılar, outputlarını sonraki işlemde input olarak kullandıkları için feedforward yapılardan ayrılmış olurlar. Recurrent ağların bir belleğe sahip olduğunu söyleyebiliriz. Bir ağa memory eklemenin sebebi ise, belli bir düzende gelen input setinin, çıktı için bir anlamı olmasıdır. Bu çeşit data setleri için feedforward ağlar yeterli olmaz.

Tam bu noktada RNN'ler devreye girer. Yazı, konuşma, zamana bağlı çeşitli sensör veya istatiksel veriler gibi belli bir sıra ile gelen verilerin yapısını anlamada recurrent ağlar kullanılır.



RNN İşlem Döngüsü

Diagramda görüldüğü gibi , RNN'in işlem döngüsünde hidden layer'dan çıkan sonuç hem output üretir, hem content unitlere yazılır. Bu şekilde, her yeni input, önceki inputların işlenmesi sonucu üretilmiş content unitlerle birlikte işlenir. Farklı zamanlarda belleğe alınan veriler arasında korelasyon bulunuyorsa buna "long term" bağımlılık denir. RNN, bu long-term bağımlılıkların arasındaki ilişkiyi

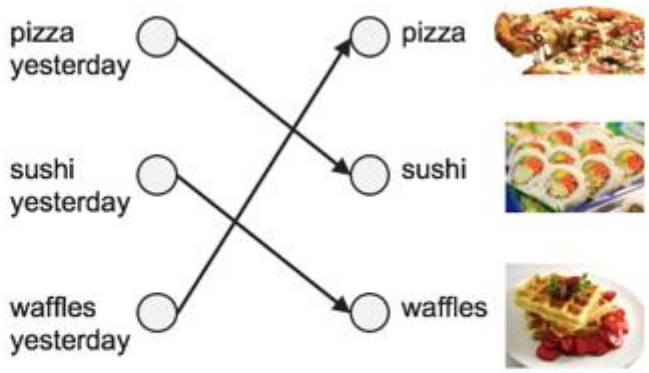
hesaplayabilen bir ağdır.

İnsanların davranışlarında, konuşma ve düşüncelerinde de bu yapıdaki gibi önceden memory'de olan bilgi tıpkı bir hidden layer'da yeni veri ile döngüye girerek işlenir. Bu işlemi yaparken kullanılan matemetiksel formül aşağıdaki gibidir.

$$\mathbf{h}_t = \phi\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right),$$

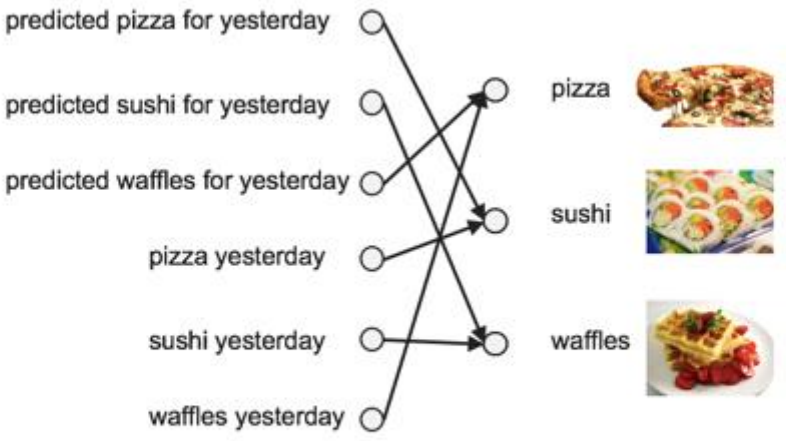
ht, t anındaki hidden layer'ın sonucudur. xt inputu W ağırlığı ile çarpılır. Daha sonra t-1 anında, content unit 'te tutulan h(t-1) değeri U ağırlığı ile çarpılır ve Wxt ile toplanır. W ve U değerleri girdi ile katman arasındaki ağırlıklardır. Burada ağırlık matrisi önceki ve şimdiki verinin hangisinin sonuca etkisi daha çok veya az ise ona göre değerler alır. Bu işlemler sonucunda oluşan hata hesaplanır ve backpropagation ile yeni ağırlık değerleri tekrar düzenlenir. Backprop işlemi hata yeterince minimize edilene kadar devam eder. Wxt + Uh(t-1) toplamı sigmoid, tanh gibi aktivasyon fonksiyonuna sokulur. Böylece çok büyük veya çok küçük değerler mantıklı bir aralığa alınır. Bu şekilde non-lineerlik de sağlanmış olur.

Örnek Bir Recurrent Neural Network



Aşçı Yemek Döngüsü

Bir aşçının 3 farklı yemeği sırasıyla yaptığını düşünelim. 1. gün pizza yaptıysa 2. gün sushi 3. gün waffle yapıyor olsun. Bir sonraki gün ne yapacağını tahmin etmemiz gerekiyorsa öncelikle nasıl bir problemle karşı karşıya olduğumuz anlayarak, bu probleme uygun bir metot kullanmamız gerekir. Burada yemek bir sıraya göre geldiği için, yani önceki gün yapılan yemeğin sonraki günde etkisi olacağı için uygulanması gereken metot recurrent neural network metodudur. Bu sayede elimizdeki bilgilerle yeni bilgi olmasa dahi birkaç hafta sonraki günde yapılacak yemeği dahi tahmin edebiliriz.

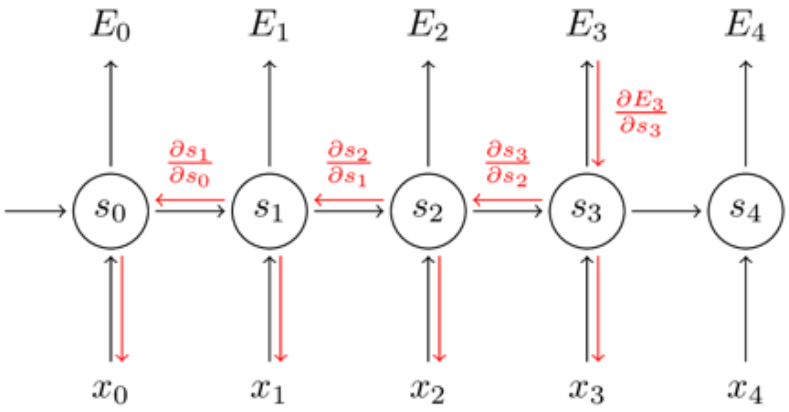


RNN yapısıyla örneğin incelenmesi

Backpropagation Trough Time(BPTT)

Recurrent ağların amacı sıralı inputları doğru bir şekilde sınıflandırmak diyebiliriz. Bu işlemleri yapabilmek için hatanın backpropunu ve gradient descentini kullanırız. Backprop, feedforward ağlarda sonda outputtaki hatayı geriye hatanın türevini ağırlıklara dağıtılarak yapılır. Bu türev kullanılarak öğrenme katsayısı, gradient descent düzenlenerek hatayı düşürecek şekilde ağırlıklar düzenlenir.

RNN için kullanılan yöntem ise BPTT diye bilinen zamana bağlı sıralı bir dizi hesaplamanın tümü için backprop uygulamasıdır. Yapay ağlar bir dizi fonksiyonu içiçe f(h(g(x))) şeklinde kullanır. Buraya zamana bağlı değişken eklendiğinde türev işlemi zincir kuralı ile çözümlenebilir.



RNN Backpropagation İşlem Döngüsü

Yukardaki RNN'in çalışma mantığı anlatılmıştır. Üstteki figürde ise sıralı 5 girdili bir recurrent ağ yapısı gösterilmektedir. E burada oluşan hatayı ifade etmektedir. Örneğin, E3 için backpropagation yaparken yaptığımız işlemde w ağırlığına göre türevi kullanılmaktadır. Bu türevi çözebilmek için zincir kuralı ile birkaç türevin çarpımını kullanırız.

$$\frac{\partial E_3}{\partial W} = \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial W}$$

$$\frac{\partial E_3}{\partial W} = \sum_{k=0}^{3} \frac{\partial E_3}{\partial \hat{y}_3}\frac{\partial \hat{y}_3}{\partial s_3}\frac{\partial s_3}{\partial s_k}\frac{\partial s_k}{\partial W}$$

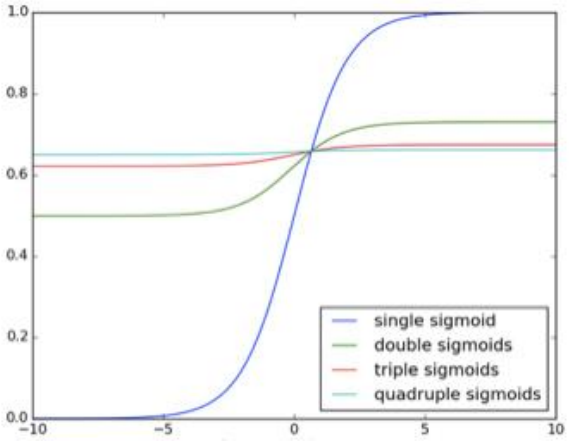
Bu çarpım yukarıda gösterildiği gibidir. Burada s3'ün açılımında s2'ye bir bağımlılık bulunur. Bunu çözebilmek için yine zincir kuralını kullanarak s3'ün s2'ye türevini de ekleyerek sonucu bulabiliriz. Böylece formüldeki gibi gradient zamana bağlı şekilde dağıtılmış olur.

Truncated BPTT

Truncated BPTT ise BPTT 'nin uzun bir sıralı veri için bir çok ileri, geri işlemler çok masraflı olacağından, BPTT için yaklaşım yapar. Bunun kötü yanı gradient kesilmiş hesaplandığından ağ "long term" bağımlılıkları full BPTT kadar iyi öğrenemez.

Vanishing/Exploding Gradient (Gradient yokolması/uçması)

Gradient tüm ağırlıkları ayarlamamızı sağlayan bir değerdir. Ancak birbirine bağlı uzun ağlarda hatanın etkisi oldukça düşerek gradient kaybolmaya başlayabilir. Bu da doğru sonucu bulmayı olanaksızlaştırır. Bütün katmanlar ve zamana bağlı adımlar birbirine çarpımla bağlı olduğundan, türevleri yokolma veya uçma yani aşırı yükselme tehlikesindedir.



Sigmoid Aktivasyon Fonksiyonunun Türeve Bağlı Değişimi

Gradient exploding yani aşırı büyümesi ağın çok büyük değerler üretmesini sağlayarak doğru sonuçtan uzaklaştıracaktır. Bunun için threshold koyarak çok yüksek değerli gradientleri kesmek basit ve etkili yollardan biridir. Gradientlerin aşırı küçülerek yok olması ise çok daha zor bir problemdir. Nerde, ne zaman durdurulması gerektiği çok açık değildir.

Neyse ki bu sorunu çözmek için de birkaç çözüm bulunmaktadır. W için uygun başlangıç değerleri seçmek yok olma etkisini azaltacaktır. Bir diğer çözüm ise sigmoid ve tanh aktivasyon fonksiyonları yerine ReLU kullanmaktır. ReLU fonksiyonunun türevi 0 veya 1'dir. Bu sebepten böyle bir problem içerisine girmeyecektir. Bir diğer yöntem ise bu problemi çözmek için dizayn edilmiş olan LSTM metodudur.