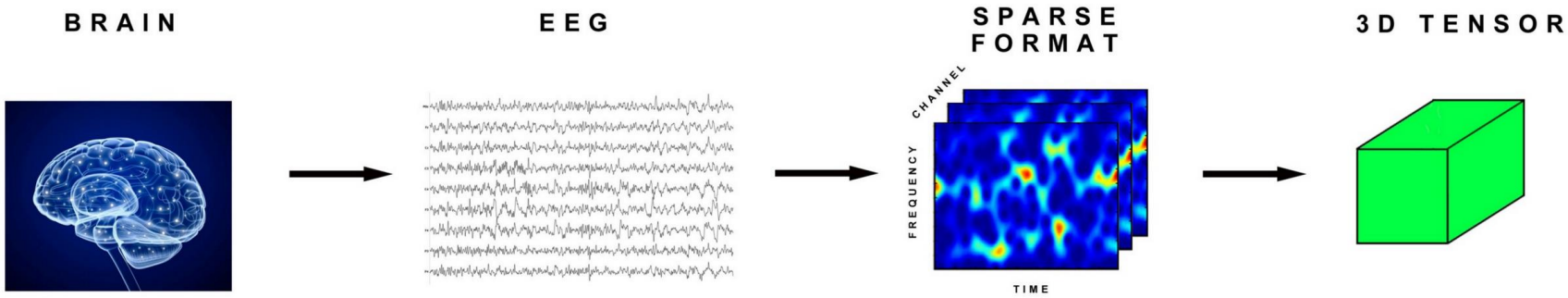
- Tensor işlemleri
- Ağ tanımlamak ve eğitmek
- Karakter tanıma örneği (MINST)

Common Data Stored in Tensors

- Vektörler ve matrislere benzer şekilde, tensörler Python'da N boyutlu dizi kullanılarak gösterilebilir.
- Çeşitli tensör türlerinde sakladığımız bazı yaygın veri kümeleri şunlardır:
- 3D = Time series
- 4D = Images
- 5D = Videos

Zaman serisi verileri (Time Series Data)

- 3D tensörler, zaman serisi verileri için çok etkilidir.
- **Tıbbi Taramalar**
  Beyinden gelen bir elektroensefalogram EEG sinyalini 3D tensör olarak kodlayabiliriz, çünkü 3 parametre olarak kapsüllenebilir:
- (time, frequency, channel)
- The transformation would look like this:

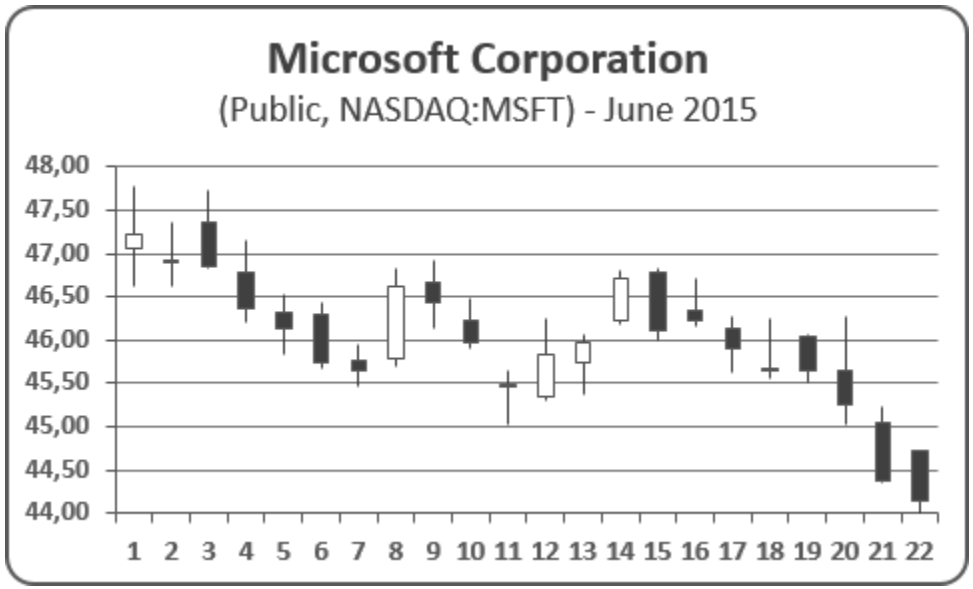


- Eğer EEG taraması olan birden fazla hastamız olsaydı, bunun gibi bir 4D tensörü olurdu:
- (sample_size, time, frequency, channel)

Kaynak:https://hackernoon.com/learning-ai-if-you-suck-at-math-p4-tensors-illustrated-with-cats-27f0002c9b32

Zaman serisi verileri (Time Series Data)

- **Stock Prices**
- Stock prices have a high, a low and a final price every minute. The New York Stock Exchange is open from 9:30 AM to 4 PM. That's 6 1/2 hours. There are 60 minutes in an hour so 6.5 x 60 = 390 minutes. These are typically represented by a candle stick graph.



Her dakika için yüksek, düşük ve nihai stok fiyatını 2D tensöründe (390,3) saklardık. Tipik bir işlem haftası için (beş gün), şu şekilde bir 3D tensörümüz olurdu:

Zaman serisi verileri (Time Series Data)

- (week_of_data, minutes, high_low_price)

- **(5,390,3)**
- 10 farklı stokumuz olsaydı, her biri bir haftalık veriye sahip olsaydı, şu şekilde olan 4D tensörümüz olurdu:

- **(10,5,390,3)**
- Şimdi 4D tensörümüz tarafından temsil edilen bir hisse senedi koleksiyonu olan ortak bir fonumuz olduğunudüşünelim. Belki de portföyümüzü temsil eden 25 adet yatırım fonu koleksiyonumuz var, şimdi 5D tensör şeklimiz var:
- **(25,10,5,390,3)**

Metin Verisi (Text Data)

- Metin verilerini de bir 3D tensörde saklayabiliriz.
- Örneğin Tweet, 140 karakterdir. Twitter, milyonlarca karakter türüne izin veren UTF-8 standardını kullanıyor, ancak temel ASCII ile aynı oldukları için, yalnızca ilk 128 karakterle ilgileniyoruz. Tek bir tweet, 2D şeklin bir vektörü (140,128) olarak kapsüllenebilir.
- Eğer 1 milyon tweet kullanırsak bunu 3D şekil tensörü olarak depolayacağız:

- **(tweet_sayisi, tweet, karakter)**

- Boyutlar:
- **(1000000,140,128)**

Görüntüler (Images)

- 4D tensörler Jpeg formatında bir dizi görüntüyü saklamak için kullanılabilir. Daha önce belirttiğimiz gibi, bir görüntü üç parametreyle saklanır:
  - Yükseklik
  - Genişlik
  - Renk derinliği
- Görüntü bir 3D tensördür, ancak görüntü seti 4D olut. Dördüncü alanın örnek numarası için kullanılır.

Görüntüler (Images)

- TensorFlow genelde görüntü verilerini aşağıdaki giib depolar:

  (sample_size, yükseklik, genişlik, color_depth)

- MNIST veri setinde 60.000 görüntü var. 28 piksel genişliğinde x 28 piksel yüksekliğindedir. Gri skalayı temsil eden 1 renk derinliğine sahiptirler.

- Bu yüzden MNIST veri setinin 4D tensörünün bir şekle sahip olduğunu söyleyebiliriz:

  (60000,28,28,1)

Renkli Görüntüler (Color Images)

- Renkli fotoğraflar, çözünürlüklerine ve kodlarına bağlı olarak farklı renk derinliğine sahip olabilir.
- Tipik bir JPG görüntüsü RGB kullanır ve böylelikle her biri kırmızı, yeşil, mavi olmak üzere 3'er renk derinliğine sahip olur.
- 750 piksel x 750 piksel görüntü için tensör boyutu:

(750,750,3)

5D Tensors

- Bir 5D tensör video verilerini saklayabilir. TensorFlow'da video verileri şöyle kodlanır:
- **(sample_size, frames, width, height, color_depth)**

- Beş dakikalık bir video çekersek (60 saniye x 5 = 300), 1920 piksel x 1080 piksel, saniyede 15 örneklenmiş renkli karede (300 saniye x 15 = 4500 kare) aşağıdaki gibi görünen bir 4D tensör depolardı:
- **(4500,1920,1080,3)**

5D Tensors

- Tensördeki beşinci alan, video setimizde birden fazla video varken devreye giriyor. Öyleyse, tam olarak 10 tane video olsaydı, 5D'lik bir şekil tensörümüz olurdu:
- **(10,4500,1920,1080,3)**

- Keras 32 bit veya 64 bit ile kayan nokta sayıları olarak saklamamızı sağlar:
  float32
  float64

- Eğer yukarıdaki 5D tensörü float32 ile saklarsak
- 10 x 4500 x 1920 x 1080 x 3 x 32
- = 8.957.952.000.000 byte
- = 1.0184 Terabyte

Tensör örnekler

- 0B Tensör (Skaler)

```
In [1]: import numpy as np

In [2]: a=np.array(3)

In [3]: a
Out[3]: array(3)

In [4]: a.ndim
Out[4]: 0
```

- 1B Tensör (Dizi)

```
In [11]: b=np.array([1,3,5,4,2])

In [12]: b.ndim
Out[12]: 1

In [13]: b
Out[13]: array([1, 3, 5, 4, 2])

In [14]: b.size
Out[14]: 5
```

Tensör örnekler

- 2B Tensör Matris

```
In [19]: c=np.array([[1,2,3],[4,5,6],[7,8,9]])

In [20]: c
Out[20]:
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

In [21]: c.ndim
Out[21]: 2

In [22]: c.size
Out[22]: 9

In [23]: c.shape
Out[23]: (3, 3)
```

Tensör örnekler

- 3B Tensör

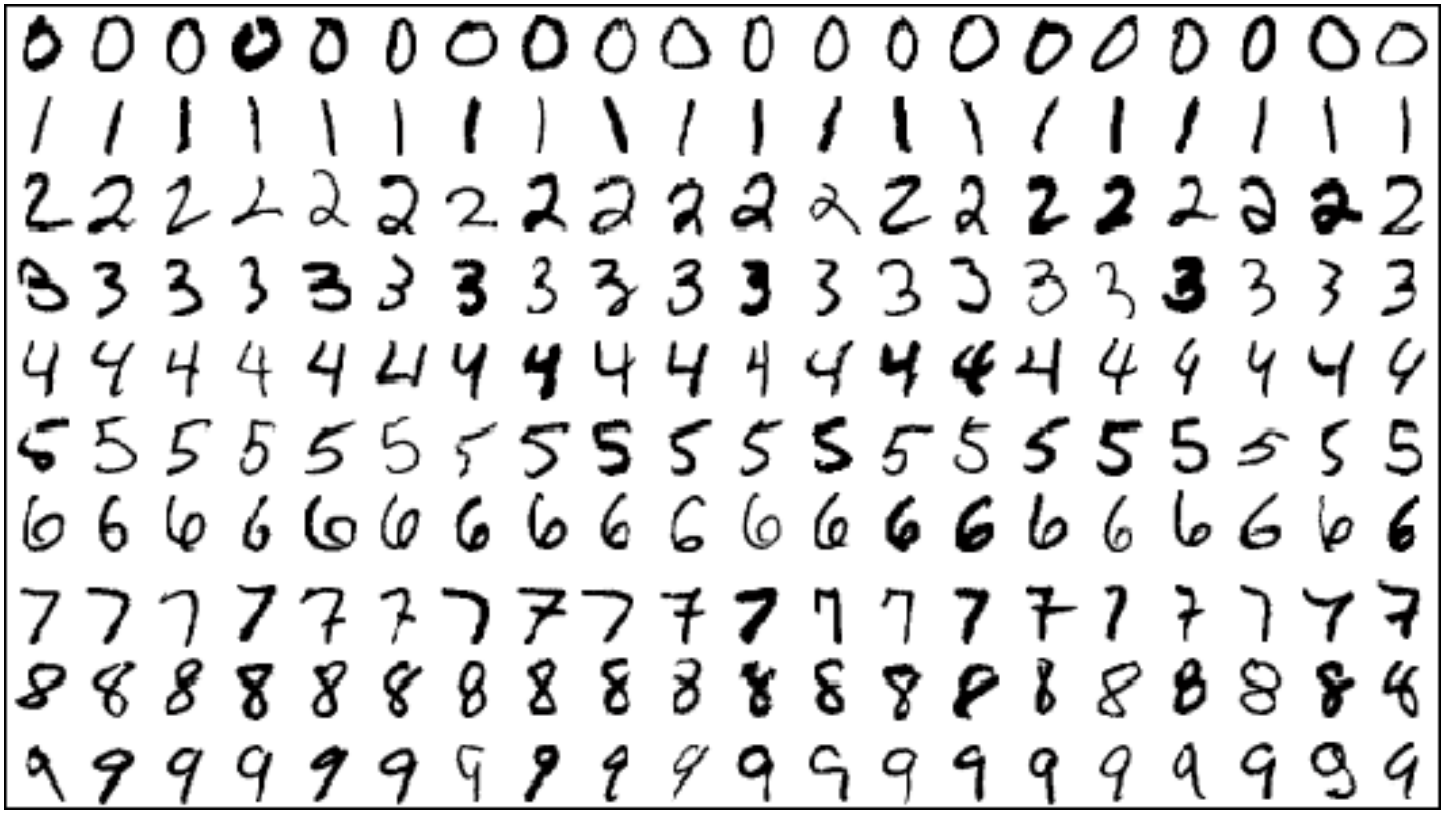
```
In [29]: d=np.random.rand(3,2,3)

In [30]: d
Out[30]:
array([[[0.81726371, 0.8597048 , 0.89624965],
        [0.03068633, 0.80458248, 0.4441645 ]],

       [[0.66730579, 0.172524  , 0.61339767],
        [0.33361243, 0.67674554, 0.05245607]],

       [[0.69444104, 0.07656473, 0.94894536],
        [0.62734666, 0.38506151, 0.5115817 ]]])

In [31]: d.ndim
Out[31]: 3

In [32]: d.shape
Out[32]: (3, 2, 3)
```

Ağ tanımlamak ve eğitmek

- Veri setini yükle: Eğitim setini ağın girişine uygun şekle dönüştürülmesi gerekebilir
- Ağ modelini seç: Sıralı (Sequential) veya Fonksiyonel (Functional) ağ yapıları
- Katmanlar ekle: Giriş ve çıkış sayıları, Katman sayısı, aktivasyon fonksiyonları
- Ağı derle: Optimizasyon fonksiyonu, loss fonksiyonu, metrikler
- Eğitimi gerçekleştir: devir sayısı (epoch), batch_size

Veri setini yüklemek

- MNIST veri seti: 0-9 arası rakamlarda oluşuyor



Veri setini yüklemek

- Keras veri setleri içerisinde , MNIST veri setini aşağıdaki komutla otomatik olarak almamızı sağlar:

```
from keras.datasets import mnist

(train_images, train_labels),
(test_images, test_labels) =
                mnist.load_data()
```

- Veri seti iki parçaya bölünmüştür:
  - Eğitim Seti (60.000 görüntü)
  - Deneme seti (10.000 görüntü)

Veri setini ağ yapısına uygunlaştırmak

- Ağın eğitiminden önce veri seti üzerinde ek işlemler yapılması gerekebilir.
- Aşağıdaki kod satırlarında reshape ile 28x28 matris formundaki görüntü 784 elemanlı bir diziye dönüştürülmüş. Ayrıca 0-255 arası değişen piksel değerleri 0-1 arasına ölçeklenmiştir.

```
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

Ağ modellleri

- Keras'ta iki ana model bulunmaktadır: *sequential* ve *functional*
- ***sequential* (sıralı) API**, çoğu problem için katmanlar halinde modeller oluşturmanıza olanak sağlar. Katmanları paylaşan veya çoklu giriş veya çıkışlara sahip modeller oluşturmanıza izin vermemesi nedeniyle sınırlıdır.
- ***functional* (işlevsel) API,** katmanların yalnızca önceki ve sonraki katmanlardan daha fazlasını bağladığı modelleri kolayca tanımlayabileceğiniz için daha fazla esnekliğe sahip modeller oluşturmasına olanak tanır.

Ağ modeli tanımlamak

- Modellerin tanımlandığı kaynak import edildikten sonra ağ modeli tanımlanır

```
from keras import models

network = models.Sequential()
```

Ağ katmanlarını tanımlamak

- Birinci katman 28*28 (görüntü boyutu) = 784 giriş,
- relu (rectified linear unit) aktivasyon fonksiyonu içeren 256 adet yapay sinirden oluşuyor.

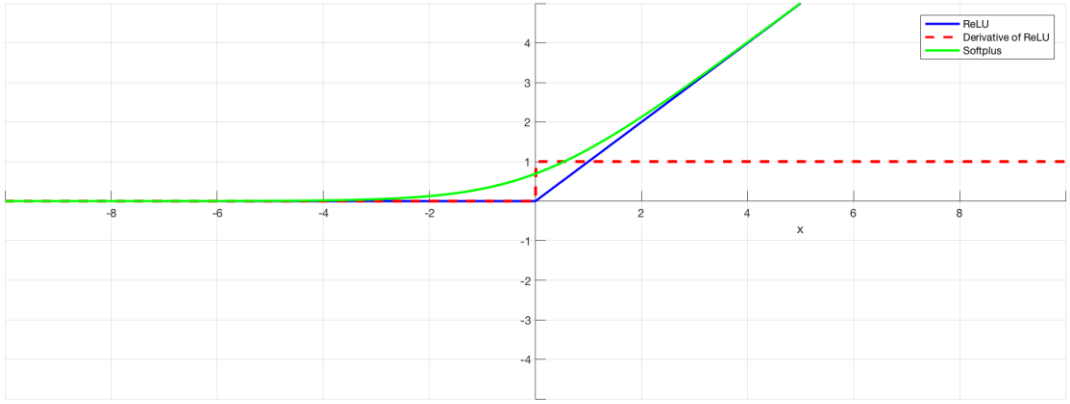
```
network.add( layers.Dense(256,
                activation='relu',
               input_shape=(28 * 28,)))
```

Ağ katmanlarını tanımlamak

Rectified Linear Units (ReLU)

Yapay sinir ağlarında ReLU gizli katman nöronlarında aktivasyon fonksiyonu olarak kullanılır. Aşağıdaki gibi tanımlanmıştır:

$$f(x)=\max(0,x)$$



Ağ katmanlarını tanımlamak

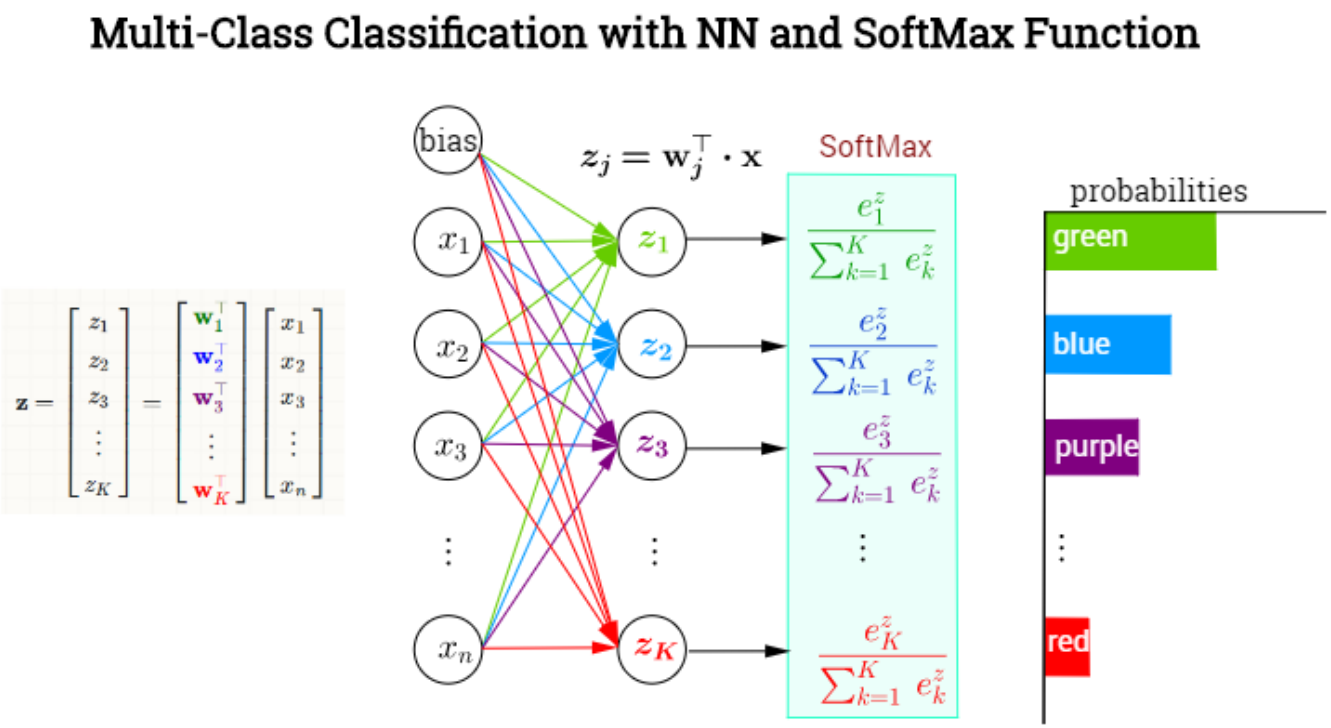
- İkinci katman ise herbiri softmax aktivasyon fonsiyonu içeren 10 adet adet yapay sinirden oluşuyor.
- Çıkış katmanı olarak görev yapan ikinci katmanda çıkış sayısı sınıf sayısına göre belirlendi.
- Yani karakter tanıma örneğinde giriş görüntüleri 10 rakamdan biri ile ilişkilendiriliyor.

```
network.add(layers.Dense(10,
                         activation='softmax'))
```

Ağ katmanlarını tanımlamak

Softmax fonksiyonunun çıktısı, **kategoriye bağlı bir dağılımı**, yani K farklı olası sonuçlara göre olasılık dağılımını temsil etmek için kullanılır.

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, j = 1, 2, \ldots, K$$



Kaynak: https://isaacchanghau.github.io/post/activation_functions/

Ağı derlemek-optimizasyon fonksiyonları

- **`optimizer`**: optimizasyon algoritması kayıp fonksiyonundaki değişime bakarak ağ parametrelerinin nasıl güncelleneceğini belirler
- https://keras.io/optimizers/
  - **SGD**
  - **Adagrad**
  - **RMSprop**:
  - **Adadelta**
  - **Adam**
  - **Adamax**
  - **Nadam**

Optimizasyon yöntemleri hakkında: https://arxiv.org/pdf/1609.04747.pdf

Ağı derlemek

- **`optimizer`**: optimizasyon algoritması kayıp fonksiyonundaki değişime bakarak ağ parametrelerinin nasıl güncelleneceğini belirler
- **`loss`**: kayıp (loss) fonksiyonu: ağın performansı loss fonksiyonu ile ölçülür
- **`metrics`**: Eğitim ve test aşamalarında gözlenecek parametreler. Burada sadece doğruluk inceleneği için accuracy seçilmiştir.

```
network.compile(
    optimizer='rmsprop',
    loss='categorical_crossentropy',
    metrics=['accuracy'])
```

Ağı derlemek - kayıp fonksiyonları

categorical_crossentropy

If your targets are one-hot encoded, use categorical_crossentropy.

Examples of one-hot encodings:

[1,0,0]
[0,1,0]
[0,0,1]

sparse_categorical_crossentropy

But if your targets are integers, use sparse_categorical_crossentropy.

Examples of integer encodings:

1
2
3

Ağı derlemek-metrikler

- **`metrics`**: Eğitim ve test aşamalarında gözlenecek parametreler.
- modelin performansını değerlendirmek için kullanılan bir fonksiyondur. Bir model derlendiğinde metrik işlevler, *metrics* parametresinde sağlanmalıdır.
- Loss fonksiyonuna benzer ancak eğitimde kullanılmaz. Kayıp işlevlerinden herhangi birini bir metrik işlev olarak kullanabilirsiniz.
  - **binary_accuracy**
  - **categorical_accuracy**
  - **sparse_categorical_accuracy**
  - **top_k_categorical_accuracy**
  - **sparse_top_k_categorical_accuracy**

```
model.compile(loss='mean_squared_error',
              optimizer='sgd',
              metrics=['mae', 'acc'])
```

```
from keras import metrics

model.compile(loss='mean_squared_error',
              optimizer='sgd',
              metrics=[metrics.mae, metrics.categorical_accuracy])
```

Ağı derlemek-metrikler

- Özel metrikler derleme adımında tanımlanabilir. Fonksiyon (y_true, y_pred) parametrelerini alır ve tek bir tensör değeri döndürür.

```
import keras.backend as K

def mean_pred(y_true, y_pred):
    return K.mean(y_pred)

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy', mean_pred])
```

Veri setini ağ yapısına uygunlaştırmak

- **`to_categorical`** ile eğitim ve test etiketleri kategorileri belirten vektörlere dönüştürülmüştür

```
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

```
In [33]: train_labels2=to_categorical(train_labels)

In [34]: train_labels[0]
Out[34]: 5

In [35]: train_labels2[0]
Out[35]: array([0., 0., 0., 0., 0., 1., 0., 0., 0., 0.], dtype=float32)

In [36]: train_labels[1]
Out[36]: 0

In [37]: train_labels2[1]
Out[37]: array([1., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)

In [38]: train_labels[2]
Out[38]: 4

In [39]: train_labels2[2]
Out[39]: array([0., 0., 0., 0., 1., 0., 0., 0., 0., 0.], dtype=float32)
```

Ağın eğitilmesi

- Derlenmiş ağ, eğitim görüntüleri ve eğitim etiketleri kullanılarak ağın eğitimi fit() fonksiyonu ile geçekleştirilebilir.
- **epochs** ile veri setinin eğitim için kaç kez tekrar edeceği belirtilir.
- **batch_size** ise bir epochs içerisinde ağırlıkların kaç örnekte bir güncelleneceğini gösterir. Eğer batch_size veri seti adedine eşit seçilirse her epoch içerisinde ağırlıklar bir kez güncellenir. Bundan dolayı bacth_size genelde yakınsama sürelerini etkiler.

```
network.fit(train_images, train_labels,
epochs=5,batch_size=128))
```

Ağın eğitilmesi

- Program çalıştırıldığında veri setini her bir dolaşımda (epoch, devir) elde edilen hata miktarları aşağıdaki gibi elde edilir.
- Eğitim sonunda %98.49 başarı elde edilmiştir.

```
Epoch 1/5
60000/60000 [==============================] - 2s 37us/step - loss: 0.2878 - acc: 0.9176
Epoch 2/5
60000/60000 [==============================] - 2s 33us/step - loss: 0.1279 - acc: 0.9631
Epoch 3/5
60000/60000 [==============================] - 2s 33us/step - loss: 0.0862 - acc: 0.9750
Epoch 4/5
60000/60000 [==============================] - 2s 33us/step - loss: 0.0642 - acc: 0.9812
Epoch 5/5
60000/60000 [==============================] - 2s 34us/step - loss: 0.0505 - acc: 0.9849

In [45]:
```

Eğitilmiş ağın performansının ölçülmesi

- Ağın test edilmesi:
- Eğitim görüntüleri dışındaki başarımı incelemek için **evaluate()** fonksiyonu ve test görüntüleri kullanılılır.
- Burada 10000 görüntü ile yapılan test sonucuna göre %97.88 başarım elde edilmiştir.
- Test ile elde edilen başarımlar genelde eğitim ile elde edilen başarımın altında kalır. Eğer test başarımı, eğitim başarımına göre çok düşükse seçilen eğitim seti örnekleri yetersiz olabilir.

```
test_loss, test_acc =
network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
print('test_loss:',test_loss)
```

```
10000/10000 [==============================] - 0s 27us/step
test_acc: 0.9788
test_loss: 0.0707745857337955 4
```

Eğitim ve test aşamaları Python dosyası

```
from keras.datasets import mnist
from keras.utils import to_categorical
from keras import models
from keras import layers

#Veri setini yükle
(train_images, train_labels),(test_images, test_labels) = mnist.load_data()

#Verileri ağ yapısına uygunlaştır
train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)

#ağ yapısını tanımla
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(28 * 28,)))
model.add(layers.Dense(10, activation='softmax'))
#Derle
model.compile(
        optimizer='rmsprop',
        loss='categorical_crossentropy',
        metrics=['accuracy'])

#Ağı train_images ile eğit
model.fit(train_images, train_labels, epochs=5, batch_size=256)

#Ağın başarımını eğitimde kullanılmamış test_images ile test et
test_loss, test_acc = model.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
print('test_loss:',test_loss)
```

Eğitilmiş ağı kaydetmek

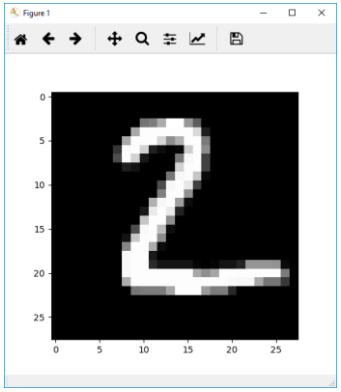
- Tüm ağı kaydetmek:

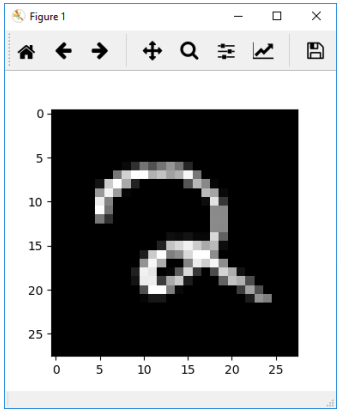
```
network.save('dosya_adi')
```

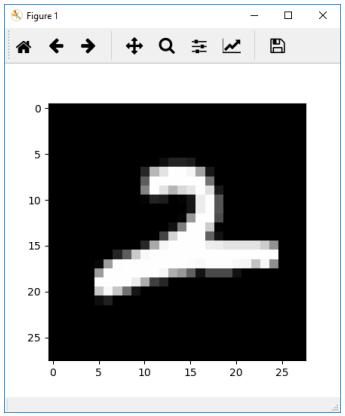
- Sadece ağırlıkları kaydetmek

```
network.save_weights('dosya_adi')
```

Eğitilmiş ağı tekrar kullanmak







Eğitilmiş ağ

2

Eğitilmiş ağı tekrar kullanmak

```
In [73]: from keras import models

In [74]: model=models.load_model('mnist.model1')

In [75]: (train_images, train_labels),(test_images, test_labels) = mnist.load_data()

In [76]: test_goruntu=test_images[0].reshape(1,28*28).astype('float32')/255

In [77]: y=model.predict(test_goruntu)

In [78]: y
Out[78]:
array([[3.7268353e-05, 9.8041175e-09, 1.0974355e-04, 5.2160835e-03,
        1.2340735e-07, 1.1374642e-05, 1.0773977e-09, 9.9419445e-01,
        3.5760764e-05, 3.9512845e-04]], dtype=float32)

In [79]: from numpy import argmax

In [80]: rakam=argmax(y)

In [81]: rakam
Out[81]: 7

In [82]: import matplotlib.pyplot as grafik

In [83]: grafik.imshow(test_images[0])
```