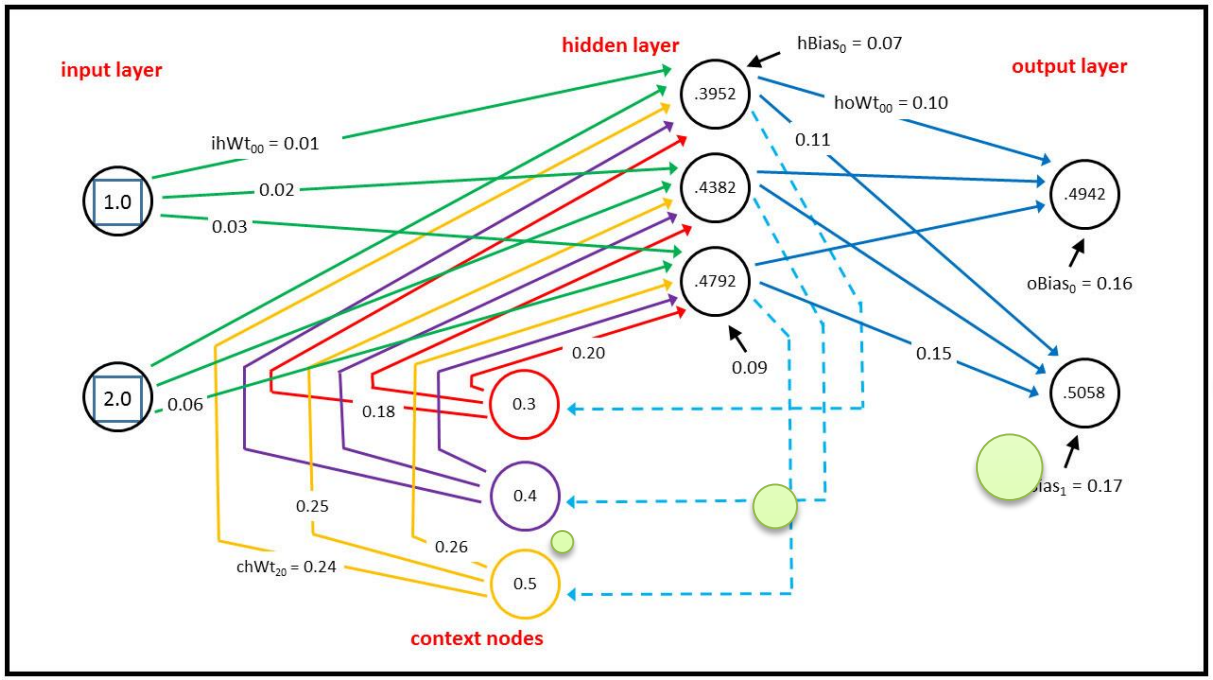
- **Recurrent Neural Networks - RNN**
- **Long Short Term Memories - LSTM**

Recurrent Neural Network- RNN

- Bir RNN'de, bilgi bir döngü boyunca ilerler. Bir karar verdiğinde, mevcut veriyi ve daha önce aldığı verilerden de ne öğrendiğini de dikkate alır.
- Normal bir RNN'nin kısa süreli bir hafızası vardır. Farklı bir RNN türü olan LSTM ise daha uzun süreli bir hafızaya sahiptir.



RNN bir önceki çıkışı tekrar giriş olarak kullanır. Geri besleme için ayrıca bir ağırlık matrisi kullanılır.

Kaynak: https://visualstudiomagazine.com/articles/2015/10/01/recurrent-neural-networks.aspx

Recurrent Neural Network- RNN

Bir katman çıkışı (Dense layer):

y_t = aktivasyon(W*input_t + b)

Bir RNN katmanı çıkışı:

y_t = aktivasyon(W*input_t + u*y_{t-1} + b)

Bir RNN katmanı çıkışı için pseudocode:

RNN katmanı verileri dizi formatında kabul eder. Aşağıda verilen input_sequence girişe uygulanan veri dizisini tanımlar. Çıkışın ilk iterasyonda hesaplanabilmesi için başlangıç durumları (state_t) belirtilmelidir. Sonraki iterasyonlarda state_t = output_t alınarak hesaplanan çıkış bir sonraki iterasyonda state_t olarak kullanılır. state_t, U ağırlıkları ile çarpılarak tekrar ağa uygulanır. Böylece daha önceki çıkışların etkisi sürdürülür.

```
state_t = 0
for input_t in input_sequence:
output_t = activation(dot(W, input_t) + dot(U, state_t) + b)
state_t = output_t
```

IMDB üzerinde Embedding ve RNN katmanı

```
 8 from keras.models import Sequential
 9 from keras.layers import Embedding, SimpleRNN
10 from keras.datasets import imdb
11 from keras.preprocessing import sequence
12 max_features = 10000
13 maxlen = 500
14
15 print('Loading data...')
16 (input_train1, y_train), (input_test, y_test) =\
17             imdb.load_data(num_words=max_features)
18
19 print(len(input_train1), 'train sequences')
20 print(len(input_test), 'test sequences')
21 print('Pad sequences (samples x time)')
22 input_train = sequence.pad_sequences(input_train1, maxlen=maxlen)
23 input_test = sequence.pad_sequences(input_test, maxlen=maxlen)
24 print('input_train shape:', input_train.shape)
25 print('input_test shape:', input_test.shape)
```

Yorumlar 500 kelime ile sınırlı. Daha az olanları sıfırlar ile 500'e tamamla

IMDB üzerinde Embedding ve RNN katmanı

```
28 from keras.layers import Dense
29
30 model = Sequential()
31 model.add(Embedding(max_features, 32))
32
33 model.add(SimpleRNN(32))
34
35 model.add(Dense(1, activation='sigmoid'))
36
37 model.compile(optimizer='rmsprop',
38               loss='binary_crossentropy',
39               metrics=['acc'])
40 history = model.fit(input_train, y_train,
41 epochs=10,
42 batch_size=128,
43 validation_split=0.2)
44 model.save('rnn_imdb1')
```

IMDB üzerinde Embedding ve RNN katmanı

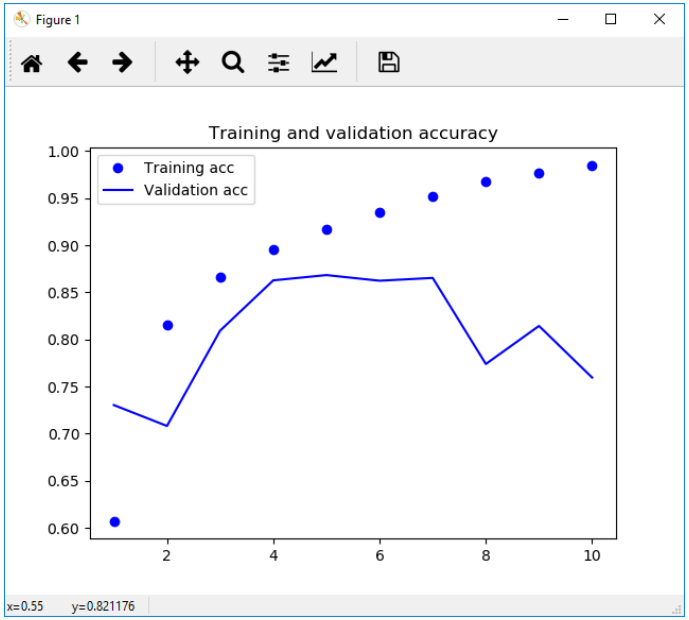
- Model.summary()

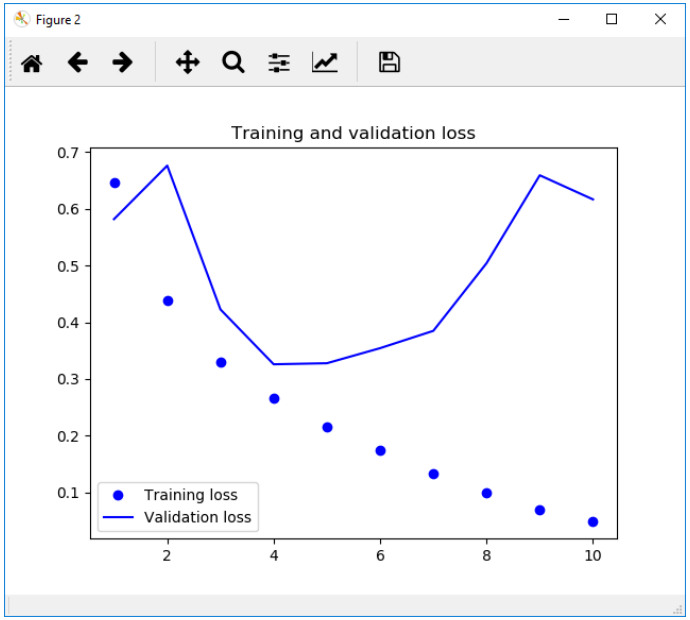
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, None, 32)          320000
_________________________________________________________________
simple_rnn_1 (SimpleRNN)     (None, 32)                2080
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 33
=================================================================
Total params: 322,113
Trainable params: 322,113
Non-trainable params: 0
```

W=32*32, U=32*32 b=32
Parametre
sayısı=32*32+32*32+32=2080

IMDB üzerinde Embedding ve RNN katmanı

```
47 import matplotlib.pyplot as plt
48
49 acc = history.history['acc']
50 val_acc = history.history['val_acc']
51 loss = history.history['loss']
52 val_loss = history.history['val_loss']
53
54 epochs = range(1, len(acc) + 1)
55 plt.plot(epochs, acc, 'bo', label='Training acc')
56 plt.plot(epochs, val_acc, 'b', label='Validation acc')
57 plt.title('Training and validation accuracy')
58 plt.legend()
59 plt.figure()
60
61 plt.plot(epochs, loss, 'bo', label='Training loss')
62 plt.plot(epochs, val_loss, 'b', label='Validation loss')
63 plt.title('Training and validation loss')
64 plt.legend()
65 plt.show()
```



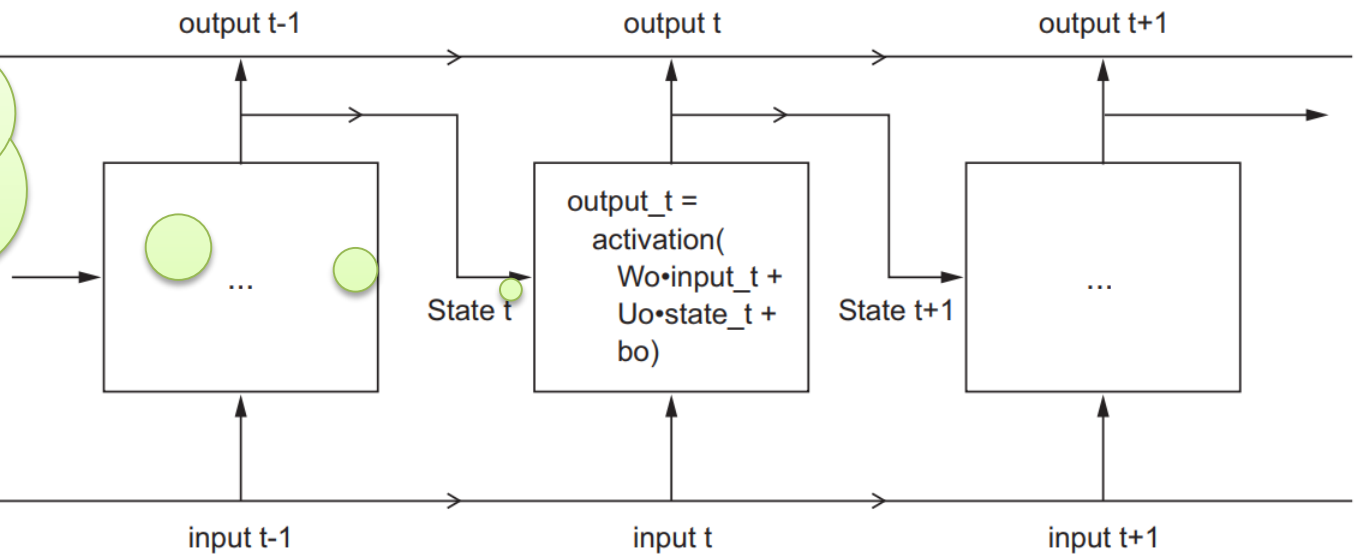


LSTM katmanı

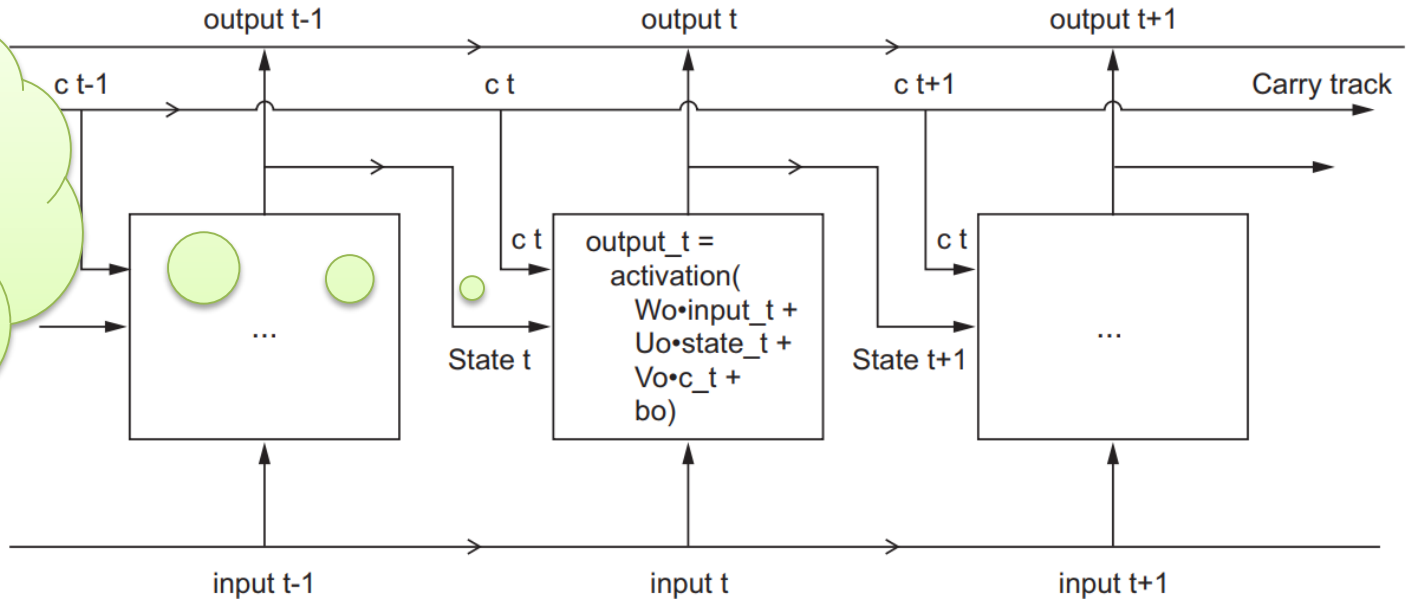
- SimpleRNN kısa zamanlı bağımlılıklaı öğrenmede faydalıdır, uzun zamanlı bağımlılıkları öğrenmede yetersiz kalır.
- Alternatif olarak uygulamada LSTM (Long Short-Term Memory) kullanılır.
- LSTM önceki adımlardaki veriyi taşımak için bir yaklaşım sunar.
- İşlenen diziye paralel çalışan bir taşıyıcı bant gibi çalışır.
- Diziden elde edilen bilgiler herhangi bir noktada taşıyıcı bandına gönderilebilir, daha sonraki bir zaman aşamasına taşınabilir ve ihtiyaç olduğunda bozulmadan geri alınabilir.
- LSTM temel olarak, daha sonra kullanılmak üzere bilgiyi saklar, böylece, eski sinyallerin işlem sırasında yavaş yavaş kaybolmasını önler.

LSTM

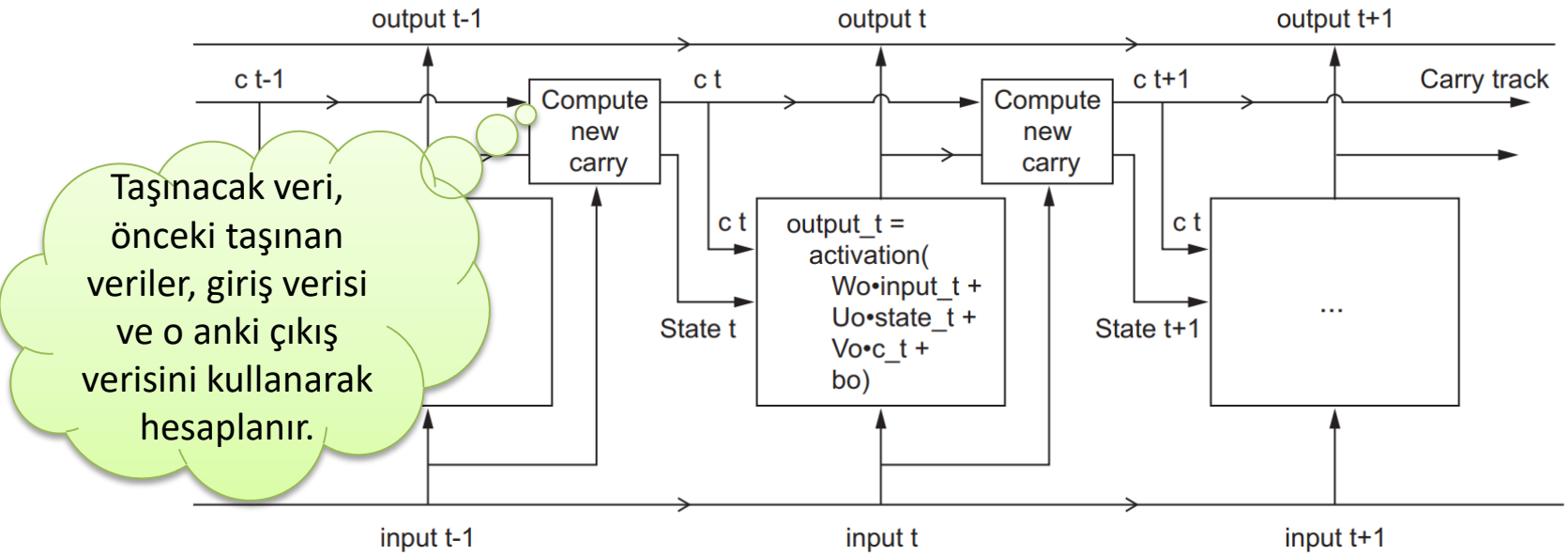
RNN önceki durumu karar vermede kullanır. Önceki durumların etkisi zamanla kaybolur.



LSTM önceki durumları ve taşıma hattını kullanarak önceki verilerin etkisini sürdürmeyi amaçlar.



LSTM



- Taşınacak verinin hesabı üç farklı dönüşüm kullanılır. Bunlar SimpleRNN hücresi yapısındadır:

```
y = activation(dot(state_t, U) + dot(input_t, W) + b)
```

LSTM

```
output_t = activation(dot(state_t, Uo)
                      + dot(input_t, Wo)
                      + dot(c_t,     Vo)
                      + bo)

i_t = activation(dot(state_t, Ui) + dot(input_t, Wi) + bi)
f_t = activation(dot(state_t, Uf) + dot(input_t, Wf) + bf)
k_t = activation(dot(state_t, Uk) + dot(input_t, Wk) + bk)

c_t+1 = i_t * k_t + c_t * f_t
```

- LSTM 4 adet RNN hücresini içerecek yapıdadır.
- c_t ve f_t'lerin çarpılması, taşıma veri akışındaki ilgisiz bilgileri unutmanın bir yoludur.
- i_t ve k_t şimdiki zaman hakkında bilgi verir, taşıma hattını yeni bilgilerle günceller.

Örnek: LSTM – Imdb

```
33 model = Sequential()
34 model.add(Embedding(max_features, 32))
35 model.add(LSTM(32))
36 model.add(Dense(1, activation='sigmoid'))
37 model.summary()
38
39 model.compile(optimizer='rmsprop',
40               loss='binary_crossentropy',
41               metrics=['acc'])
42
43 history = model.fit(input_train, y_train,
44                     epochs=10,
45                     batch_size=128,
46                     validation_split=0.2)
```

Önceki örnekte SimpleRNN yerine LSTM kullanılarak eğitim gerçekleştiriliyor.

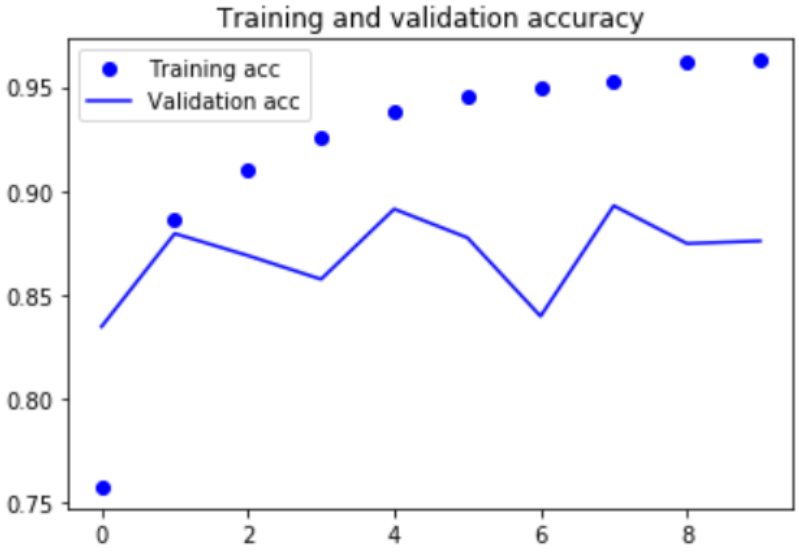
LSTM – Imdb

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_19 (Embedding)     (None, None, 32)          320000
_________________________________________________________________
lstm_2 (LSTM)                (None, 32)                8320
_________________________________________________________________
dense_25 (Dense)             (None, 1)                 33
=================================================================
Total params: 328,353
Trainable params: 328,353
Non-trainable params: 0
_________________________________________________________________
```

Bir LSTM katmanında dört Simple RNN var gibi düşünülebilir. Verilen örnekte, 32 girişli ve 32 çıkışlı bir LSTM 32*32 giriş ağırlıkları, 32*32 geri besleme ve her hücre için bir biasla birlikte 32*32+32*32+32 adet eğitilecek ağırlık bulunmakta. Toplamda (32*32+32*32+32)*4=8320

LSTM

- IMDB problemi gibi kullanılan kelimeler ve frekanslarına bakılarak çözülebilecek problemlerde tam bağlantılı ağ ile benzer sonuçlar elde edilebilir.
- Ancak soru cevaplama ve makine tercümesi gibi problemlerde LSTM daha başarılı sonuçlar üretir.



Örnek: Sıcaklık tahmini

```
19 import os
20 data_dir = 'jena_climate'
21 fname = os.path.join(data_dir, 'jena_climate_2009_2016.csv')
22 f = open(fname)
23 data = f.read()
24 f.close()
25 lines = data.split('\n')
```

Jena_climate veri seti 2009–2016 yılları arasında 10dk aralıklarla alınmış 420551 satır, sıcaklık, nem basınç, rüzgar yönü gibi 14 çeşit ölçümü içeriyor.

```
In [10]: lines[0]
Out[10]: '"Date Time","p (mbar)","T (degC)","Tpot (K)","Tdew
(degC)","rh (%)","VPmax (mbar)","VPact (mbar)","VPdef (mbar)","sh
(g/kg)","H2OC (mmol/mol)","rho (g/m**3)","wv (m/s)","max. wv (m/
s)","wd (deg)"'

In [11]: lines[1]
Out[11]: '01.01.2009
00:10:00,996.52,-8.02,265.40,-8.90,93.30,3.33,3.11,0.22,1.94,3.12,1
307.75,1.03,1.75,152.30'

In [12]: lines[2]
Out[12]: '01.01.2009
00:20:00,996.57,-8.41,265.01,-9.28,93.40,3.23,3.02,0.21,1.89,3.03,1
309.80,0.72,1.50,136.10'

In [13]: lines[3]
Out[13]: '01.01.2009
00:30:00,996.53,-8.51,264.91,-9.31,93.90,3.21,3.01,0.20,1.88,3.02,1
310.24,0.19,0.63,171.60'

In [14]: lines[4]
Out[14]: '01.01.2009
00:40:00,996.51,-8.31,265.12,-9.07,94.20,3.26,3.07,0.19,1.92,3.08,1
309.19,0.34,0.50,198.00'

In [15]: lines[5]
Out[15]: '01.01.2009
00:50:00,996.51,-8.27,265.15,-9.04,94.10,3.27,3.08,0.19,1.92,3.09,1
309.00,0.32,0.63,214.30'

In [16]: lines[6]
Out[16]: '01.01.2009
01:00:00,996.50,-8.05,265.38,-8.78,94.40,3.33,3.14,0.19,1.96,3.15,1
307.86,0.21,0.63,192.70'
```

Örnek: Sıcaklık tahmini

```
19 import os
20 data_dir = 'jena_climate'
21 fname = os.path.join(data_dir, 'jena_climate_2009_2016.csv')
22 f = open(fname)
23 data = f.read()
24 f.close()
25 lines = data.split('\n')
26
27 header = lines[0].split(',')
28 lines = lines[1:]
29 print(header)
30 print(len(lines))
31
32
33 import numpy as np
34 float_data = np.zeros((len(lines), len(header) - 1))
35
36 for i, line in enumerate(lines):
37     values = [float(x) for x in line.split(',')[1:]]
38     float_data[i, :] = values
39
```

Verileri işlemeden önce dosyadan okuyup 420551x14 elemanlı bir Numpy dizisine dönüştürdük.

```
Console 1/A                                                    02:54:37

In [58]: float_data.shape
Out[58]:

(420551,    14)


In [59]: float_data[0]
Out[59]:
array([ 9.96520e+02, -8.02000e+00,  2.65400e+02,
-8.90000e+00,
        9.33000e+01,  3.33000e+00,  3.11000e+00,
2.20000e-01,
        1.94000e+00,  3.12000e+00,  1.30775e+03,
1.03000e+00,
        1.75000e+00,  1.52300e+02])

In [60]: float_data[1]
Out[60]:
array([ 9.9657e+02, -8.4100e+00,  2.6501e+02,
-9.2800e+00,  9.3400e+01,
        3.2300e+00,  3.0200e+00,  2.1000e-01,
1.8900e+00,  3.0300e+00,
        1.3098e+03,  7.2000e-01,  1.5000e+00,
1.3610e+02])
```

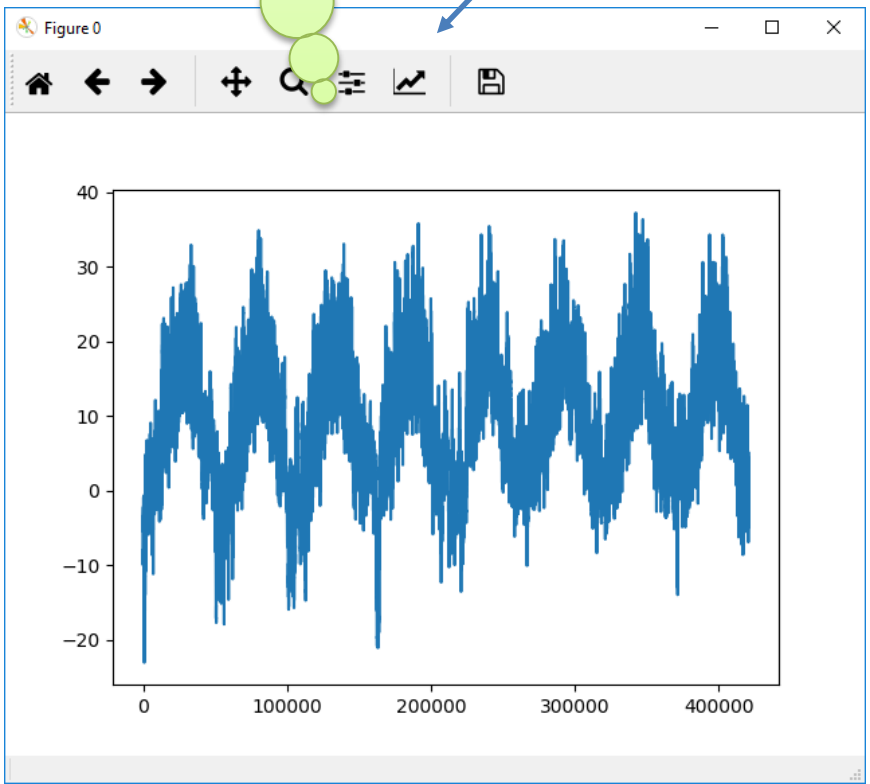
Dosyadaki rakamları ayırmak için virgülü kullan

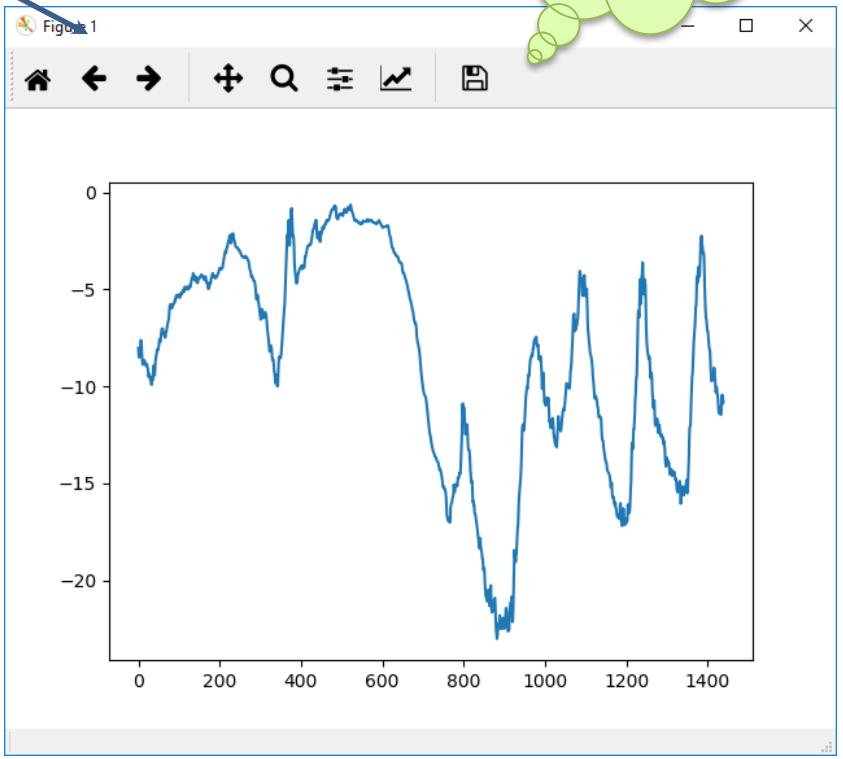
Örnek: Sıcaklık tahmini

Sıcaklığın zamana bağlı değişiminin grafiği yıllara göre periyodik davranış sergilediği görülmektedir.

```
41 from matplotlib import pyplot as plt
42 #örnek çizimler
43 temp = float_data[:, 1] # temperature (in degrees Celsius)
44 plt.figure(0)
45 plt.plot(range(len(temp)), temp)
46 plt.figure(1)
47 plt.plot(range(1440), temp[:1440])
48
```

10 günlük sıcaklık verisi



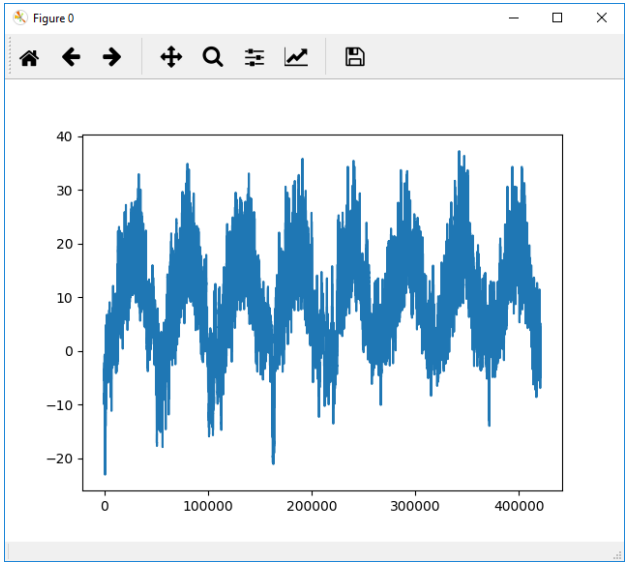


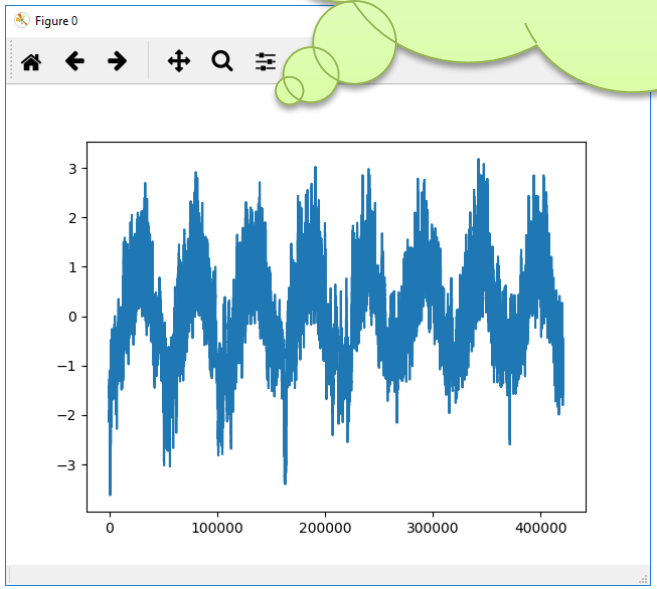
Örnek: Sıcaklık tahmini -Verilerin hazırlanması

- lookback = 720, i.e. our observations will go back 5 days.
- steps = 6, i.e. our observations will be sampled at one data point per hour.
- delay = 144, i.e. our targets will be 24 hours in the future.

```
49 # verilerin normalleştirilmesi
50 mean = float_data[:200000].mean(axis=0)
51 float_data -= mean
52 std = float_data[:200000].std(axis=0)
53 float_data /= std
```

Verilerden ortalaması çıkartılırsa sıfır civarında değişmesi sağlanır. Standart sapmaya bölünürse işaretin minimum ve maksimum değerleri sınırlanmış olur.





Örnek: Sıcaklık tahmini -Verilerin hazırlanması

```
56 def generator(data, lookback, delay, min_index, max_index,
57                shuffle=False, batch_size=128, step=6):
58
59     if max_index is None:
60         max_index = len(data) - delay - 1
61
62     i = min_index + lookback
63
64     while 1:
65         if shuffle:
66             rows = np.random.randint(min_index + lookback,
67                                      max_index,
68                                      size=batch_size)
69         else:
70             if i + batch_size >= max_index:
71                 i = min_index + lookback
72             rows = np.arange(i, min(i + batch_size, max_index))
73             i += len(rows)
74
75         samples = np.zeros((len(rows),
76                             lookback // step,
77                             data.shape[-1]))
78         targets = np.zeros((len(rows),))
79
80         for j, row in enumerate(rows):
81             indices = range(rows[j] - lookback, rows[j], step)
82             samples[j] = data[indices]
83             targets[j] = data[rows[j] + delay][1]
84         yield samples, targets
```

generator() daha önce kullandığımız ImageDataGenerator class'a benzer şekilde benzer şekilde kullanılacaktır. Veriler step ile belirtilen 6 örnekte bir, toplam batch_size=128 örnek olacak şekilde fit_generator'e sağlanacaktır.

data—The original array of floating-point data, which you normalized.
lookback—How many timesteps back the input data should go.
delay—How many timesteps in the future the target should be.
min_index and max_index —Indices in the data array that delimit which timesteps to draw from. This is useful for keeping a segment of the data for validation and another for testing.
shuffle—Whether to shuffle the samples or draw them in chronological order.
batch_size —The number of samples per batch.
step —The period, in timesteps, at which you sample data. You'll set it to 6 in order to draw one data point every hour.

Örnek: Sıcaklık tahmini -Verilerin hazırlanması

```
lookback = 1440 # son 10 günlük veri
step = 6 # 1 saat ara ile örnekle
delay = 144 # 24*6 1 günlük gecikme
batch_size = 128

train_gen = generator(float_data,
                      lookback=lookback,
                      delay=delay,
                      min_index=0,
                      max_index=200000,
                      shuffle=True,
                      step=step,
                      batch_size=batch_size)

val_gen = generator(float_data,
                    lookback=lookback,
                    delay=delay,
                    min_index=200001,
                    max_index=300000,
                    step=step,
                    batch_size=batch_size)

test_gen = generator(float_data,
                     lookback=lookback,
                     delay=delay,
                     min_index=300001,
                     max_index=None,
                     step=step,
                     batch_size=batch_size)
```

- Tanımladığımız generator() fonksiyonu kullanılarak eğitim, geçerleme ve test için birer generator tanımlanır. Burada 200000 örnek eğitim, 100000 geçerleme ve geri kalanı da test amaçlı kullanılmıştır.
- lookback=1440 ile son 10 günlük veri kullanılarak delay=144 ile 1 günlük hava tahmini yapılmıştır.

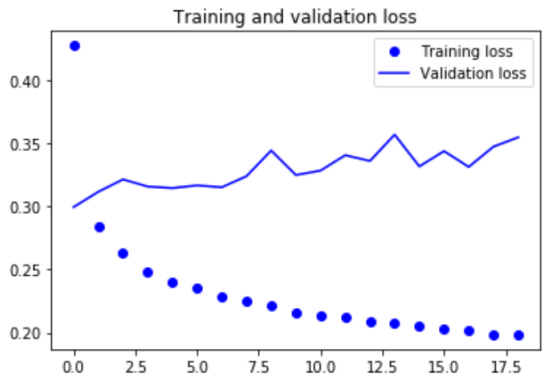
Örnek: Sıcaklık tahmini -Verilerin hazırlanması

```
146 from keras.models import Sequential
147 from keras import layers
148 from keras.optimizers import RMSprop
149
150 model = Sequential()
151 model.add(layers.Flatten(input_shape=(lookback // step,
152                                       float_data.shape[-1])))
153
154 model.add(layers.Dense(32, activation='relu'))
155
156 model.add(layers.Dense(1))
157
158 model.compile(optimizer=RMSprop(),
159               loss='mae',
160               metrics=['acc'])
161
162 history = model.fit_generator(train_gen,
163                               steps_per_epoch=500,
164                               epochs=20,
165                               validation_data=val_gen,
166                               validation_steps=val_steps)
167
```

lookback // step=1440/6=240 float_data.shape[]-1=14

Temel makine öğrenmesi yaklaşımı ile eğitildiğinde aşağıdaki gibi geçerleme başarımı mae cinsinde 0.30-0.35 civarında elde edildi. Bir miktar aşırı uydurma da ortaya çıktı.

```
170 import matplotlib.pyplot as plt
171 loss = history.history['loss']
172 val_loss = history.history['val_loss']
173 epochs = range(len(loss))
174 plt.figure()
175 plt.plot(epochs, loss, 'bo', label='Training loss')
176 plt.plot(epochs, val_loss, 'b', label='Validation loss')
177 plt.title('Training and validation loss')
178 plt.legend()
179 plt.show()
```



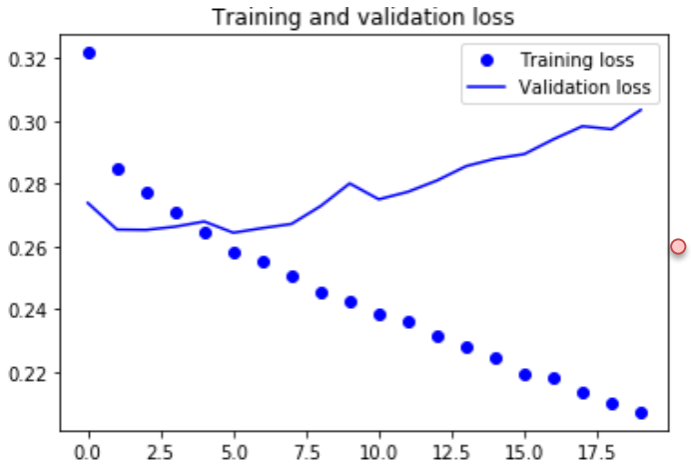
Örnek: Sıcaklık tahmini -Verilerin hazırlanması

```
model = Sequential()
model.add(layers.GRU(32, input_shape=(None,
                                      float_data.shape[-1])))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(),
              loss='mae')

history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=20,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```

- Önceki uygulamadaki ilk dense layer yerine LSTM'nin daha basit bir çeşidi olan GRU (Gated Recurrence Unit) katmanı kullanıyoruz.
- GRU katmanları, LSTM ile benzer prensibe göre çalışır.
- Daha az hesaplama gücü gerektirmesine rağmen, LSTM kadar güçlü bir model sağlamaz.
- GRU Ek bilgi: https://arxiv.org/pdf/1412.3555.pdf

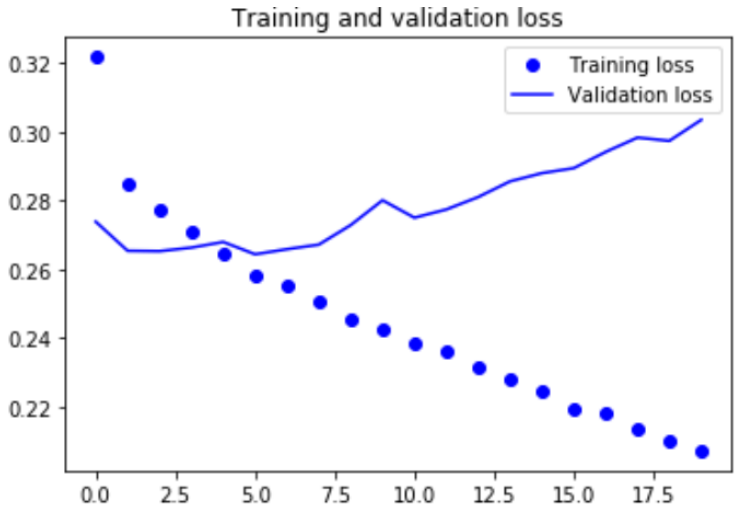


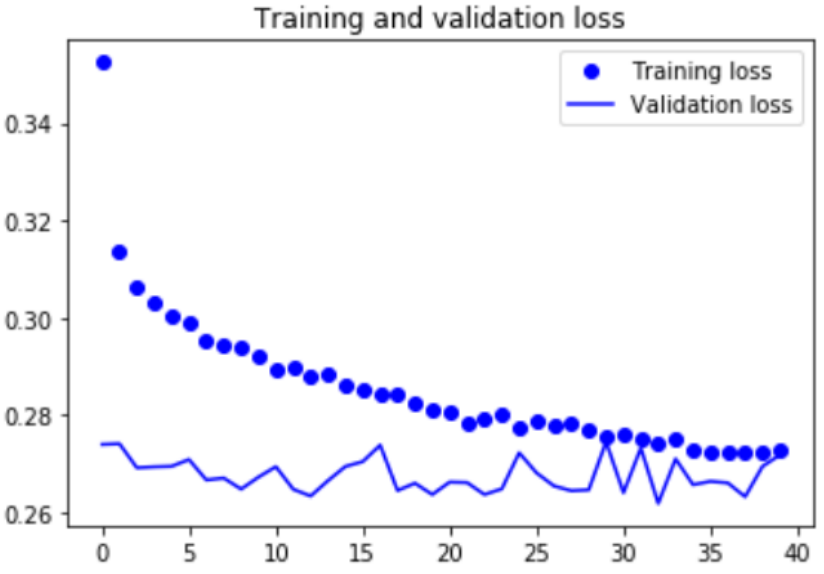
GRU ile geçerleme başarımı mae cinsinden 0.27-0.30 civarına düştü. Öncekine göre düşük olsa da yine uydurma ortaya çıktı.

recurrent dropout

```
151 model = Sequential()
152 #model.add(layers.GRU(32, input_shape=(None,
153 #                                       float_data.shape[
154
155 model.add(layers.GRU(32,
156                      dropout=0.2,
157                      recurrent_dropout=0.2,
158                      input_shape=(None, float_data.shape[-1])))
159
160 model.add(layers.Dense(1))
161
```

Dense layer'a benzer şekilde dropout eklemenin yanında, geri besleme için kullanılan çıkışlara da recurrent_dropout ile dropout tanımlayabiliriz.





RNN katmanlarının artarda kullanılması (Stacking recurrent layers)

- RNN katmanları giriş olarak artarda gelen veri dizisi kabul eder.
- Bir RNN katmanının çıkışı başka bir RNN katmanına bağlanacaksa çıkışın dizi formatında olması için return_sequences=True olmalıdır.

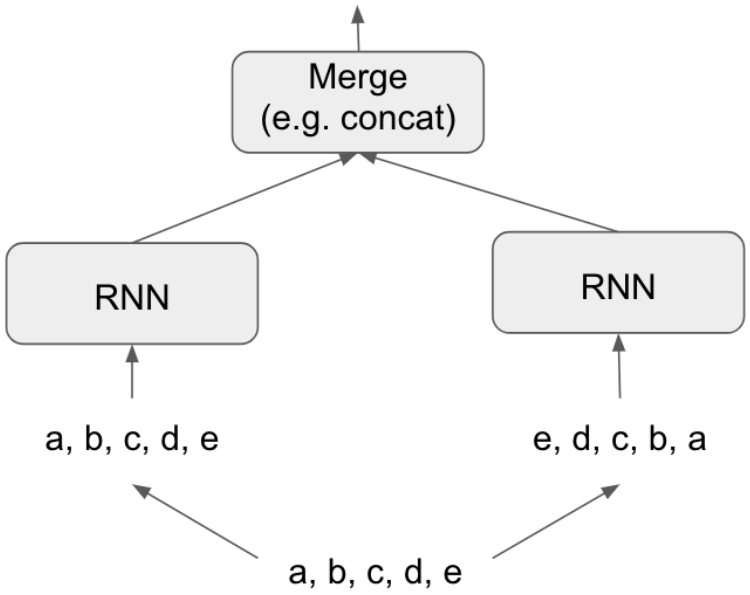
Örnek:

```
model = Sequential()
model.add(layers.GRU(32,
                     dropout=0.1,
                     recurrent_dropout=0.5,
                     return_sequences=True,
                     input_shape=(None, float_data.shape[-1])))
model.add(layers.GRU(64, activation='relu',
                     dropout=0.1,
                     recurrent_dropout=0.5))
model.add(layers.Dense(1))
```

Örnek:

```
model = Sequential()
model.add(Embedding(10000, 32))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32, return_sequences=True))
model.add(SimpleRNN(32))
```

Çift yönlü (Bidirectional) RNN



- Çift yönlü RNN, giriş dizilerine iki yönlü bakar.
- Böylece zamana bağlı ardışıl gelen dizileri ters sırada da değerlendirerek, verilerin tek yönde gelmesinden olayı kaybedilmiş olabilecek özelliklerin de iyileştirilmesini sağlar.
- Çift yönlü RNN, Bidirectional katman eklenerek kullanılır.
- Örnek:
  model.add(layers.**Bidirectional**(layers.**LSTM**(32)
  input_shape=(None, float_data.shape[-1])))
- model.add(layers.**Bidirectional**(
  layers.**GRU**(32), input_shape=(None,
  float_data.shape[-1])))

Kaynak: https://github.com/fchollet/deep-learning-with-python-notebooks/blob/master/6.3-advanced-usage-of-recurrent-neural-networks.ipynb

Çift yönlü (Bidirectional) RNN

```
from keras.models import Sequential
from keras import layers
from keras.optimizers import RMSprop

model = Sequential()
model.add(layers.Bidirectional(
    layers.GRU(32), input_shape=(None, float_data.shape[-1])))
model.add(layers.Dense(1))

model.compile(optimizer=RMSprop(), loss='mae')
history = model.fit_generator(train_gen,
                              steps_per_epoch=500,
                              epochs=40,
                              validation_data=val_gen,
                              validation_steps=val_steps)
```