# LSTM Algoritması ile Hava Durumu Tahmini Üretimi

Ozan Köyük - N20230337
*Hacettepe Üniversitesi Veri ve Bilgi Mühendisliği Yüksek Lisans Programı*
*VBM 683 Makine Öğrenmesi Dersi*

***Özet*—Bu dökümanda, Ankara ilinin yakın zamandaki hava durumu verilerini, bir Yinelemeli Sinir Ağı (Recurrent Neural Networks, RNN) olan Uzun Kısa Süreli Bellek (Long-Short Term Memory, LSTM) algoritması ile birlikte kullanarak ileri tarihli hava sıcaklığı tahmini üretimi için gerekli adımlar ve açıklamalar vardır.**

## I. GIRIS

Özet kısmında da belirtildiği gibi, projenin amacı, mevcutta var olan bir veri seti ve LSTM algoritmasını kullanarak gelecek tarihli bir tahmin verisi üretmek.

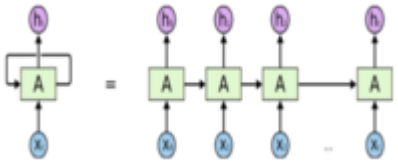
## II. KULLANILAN ARAÇLAR

Projenin tamamı Python (3.8.10) dili kullanılarak geliştirilmiştir. Kullanılan temel kütüphanelerden bazıları şunlardır: ***Keras, Scipy, Matplotlib, Pandas, Sklearn***. Verilerin tamamı, açık bir şekilde sunulan bir web sitesi aracılığı ile CSV formatında indirilip işlenmiştir [1].

Sanal ortam (virtual environment) olarak Python'a ait *venv* ortamı kullanılmıştır.

## III. LSTM ALGORITMASI

LSTM algoritması, 1997 yılında Hochreiter & Schmidhuber tarafından bulunmuştur [2]. LSTM algoritması tekrarlanan bir sinir ağı algoritmasıdır (RNN). Her hücre, kendinden bir sonraki hücreye bilgi/tecrübe aktarır.

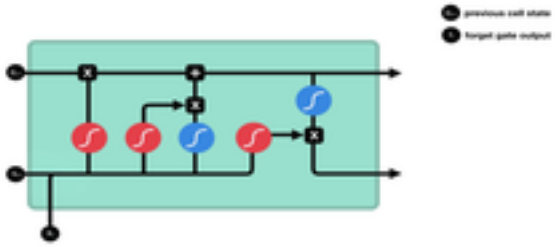



Şekil 1. RNN algoritmalarının işleme mantığı

Her bir hücre, içinde 4 veya daha fazla farklı kapı bulunmaktadır.

### A. Unutma Kapısı (Forget Gate)

Önceki iterasyondan/hücreden gelen veri ile sıradaki iterasyona/hücreye giren veri, ilk sigmoid fonksiyonu olan unutma kapısı'na girer. Burada sigmoid fonksiyonu, giren verileri 0 ve 1 arasında sıkıştırır. Bu dağılım sonucunda 0 değerine yakın olan veriler unutulur, 1 değerine yakın veriler ise bir sonraki iteresyona aktarılır.



Şekil 2. LSTM algoritmasına ait bir hücre

### B. Girdi Kapısı (Input Gate)

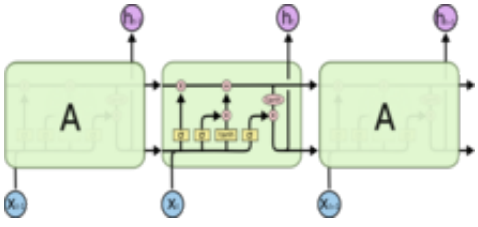
Hücrenin durumunu güncellemek için bu kapı kullanılır. Sigmoid fonksiyonuna önceki durumdan gelen veri ve şu anki girdi verileri gönderilir. 0 ile 1 arasında veri dağılımı yapılır (1 önemli, 0 önemsiz). Daha sonra veriler Tanh ('relu' gibi farklı fonksiyonlar kullanılabilir) fonksiyonuna gönderilip -1 ile 1 arasında dağılım yapılır, daha sonra çıkan sonuç ile önceki sigmoid fonksiyonundan gelen sonuçlar çarpılır.

### C. Hücre Durumu (Cell State)

Bu kapının görevi, bilgi transferini sağlamaktır. Hücre içi ve hücreler arası transferi sağlar.

### D. Çıktı Kapısı (Output Gate)

Tüm işlemler tamamlandıktan sonra, bir sonraki hücreye/katmana gönderilecek veriye karar veren kapı Çıktı Kapısıdır. Tahmin işlemi burada yapılır. Tanh fonksiyonundan çıkan veri ile bir önceki sigmoid fonksiyonundan çıkan veri çarpılır ve bir sonraki hücreye iletilir. Bu şekilde bir sonraki




Şekil 3. LSTM algoritmasına ait yapı

hücreye veri transferi yapılmış olunup, bu hücrede

elde edilen tecrübeler, bir sonraki hücreye transfer edilmiş olunur.

## IV. Kullanılan Veri

Tahmin üretimi için gerekli veri seti, ilgili bağlantıdan [1] API aracılığı ile indirilip CSV formatında saklanmıştır. Dosya içerisinde aşağıdaki sütunlar bulunmaktadır.

| Sütun Adı | Örnek Değer |
|---|---|
| location | Ankara/Esenboga |
| datetime | 2019-01-01T00:20:00.000Z |
| temp | 27 |
| dew point | 21 |
| humidity | 80 |
| wind | ENE |
| wind speed | 5 |
| wind gust | - |
| pressure | 26 |
| condition | Mostly Cloudly |

Tablo I
VERI SETI VE KOLONLAR

Veri setinde toplam 68012 adet veri bulunmaktadır. Her 2 veri, 1 saate denk gelmektedir, 2 veri arasında 30 dakikalık fark vardır. 1 Ocak 2019 tarihi ile raporun yazıldığı tarih olan 10 Aralık 2021 tarihleri arasında veri bulunmaktadır. İlgili web sitesinde Ankara için birçok ölçüm noktası mevcuttur, bu yüzden veriler en çok kullanılan lokasyon tarafından çekilmiştir.

## V. Çalıştırmadan Önce

### A. Kütüphaneler

Kullandığım kütüphaneler ve sürümleri aşağıdaki tabloda gösterilmiştir.

| Kütüphane | Versiyon |
|---|---|
| Tensorflow | 2.7.0 |
| Scipy | 1.7.3 |
| Numpy | 1.21.4 |
| Jupyter Client | 7.1.0 |
| Jupyter Core | 4.9.1 |
| Pandas | 1.3.4 |
| Matplotlib | 3.5.0 |
| Keras | 2.7.0 |
| Python | 3.8.10 |

Tablo II
PROJEDE KULLANILAN TEMEL KÜTÜPHANELER VE SÜRÜMLERI.

Jupyter ile ilgili kütüphaneler opsiyonel olarak eklenmiştir. Bir IDE aracılığı ile Jupyter bağlantısı sağlayarak projemi geliştirdim.

### B. Çalıştırmadan Önce Yapılması Gerekenler

Projeyi indirdikten sonra, ilk olarak bir sanal ortam (virtualenv) oluşturulması gerekmektedir. Aşağıdaki komut ile Python3 kullanarak bir sanal ortam oluşturuyoruz.

```
python3 -m venv venv
. venv/bin/activate
```

Bu şekilde, oluşturduğumuz virtualenv içine girip aktif hale getirdik. Ardından projenin içinde bulunan "requirements.txt" dosyasını kullanarak ilgili tüm kütüphanelerin kurulumunu yapıyoruz.

```
pip install -r requirements.txt
```

Böylece ilgili tüm kütüphanelerin kurulumunu tamamlamış olduk.

### C. Çalıştırma

Gerekli tüm kütüphanelerin kurulumlarını tamamladıktan sonra, hala virtual environment içindeyken Python script'ini çalıştırıyoruz.

```
python3 lstm_prediction.py
```

Terminal ile kodu çalıştırdığımızda, işlem sonucunda ekranda gösterilmesi gereken grafikleri maalesef göremeyeceğiz. Bunun yerine sadece kodun terminale yazdıracağı çıktıları göreceğiz. Fakat kod, oluşturduğu grafikleri, bulunan dizin içerisini belirli bir formatta kayıt ediyor. Böylece terminalden çalıştırıldığında bile, çıktı olarak ilgili grafikleri görüntüleyebiliyoruz.

Kod çalışmaya başlayınca, aşağıdakine benzer bir çıktı görüyoruz. Her bir iterasyon ile ilgili bilgileri buradan takip edebiliyoruz.

```
1/6796 [=.....................]
ETA: 3:11
32ms/step
loss: 0.0013
```

## VI. Kodlama ve Açıklama

Kodun tamamını raporumda açıklamayacağım. Bunun yerine raporumun bu kısmında kısaca algoritma ve parametereler hakkında sahte kod (pseudocode) şeklinde bilgilendirme yapacağım.

- Öncelikle CSV dosyası Pandas kütüphanesi aracılığı ile okunup parse edilir.
- İhtiyaç duyulan sütunlar alınır ve Fahrenheit cinsi olan değerler, Celcius tipine çevrilir.
- Sıcaklık değerlerinin dönüşümü tamamlandıktan sonra, önce Median filtresi ve ardından Gaussian filtresi uygulanır. Böylece veri üzerinde bir miktar smoothing uygulanmış olur.
- Train ve test için veri üzerinde paylaşım yapılır. Bu paylaşım %80'e %20 olarak belirlenmiştir.
- MinMaxScaler fonksiyonu ile, sıcaklık değerleri 0 ile 1 arasındaki değerlere dönüştürülür. Ardından önceden belirnemiş olan `TIMESTAMP` değeri ile split edilir. Bu şekilde elimizde içinde birçok array olan bir array elde etmiş oluyoruz. Bu arraylerin içinde `TIMESTAMP` değeri kadar ardaşık elemanlar bulunur.

- `Sequential` modelini oluşturulur, böylece model içerisine birden fazla `LSTM` hücresi eklenebilir hale getirilir.
- Projede 3 hücreli bir yapı kullanılmıştır. İlk `LSTM` hücresini oluştururken içerisine kullanılacak algoritmayı ('tanh', 'relu',..) vererek oluşturulur. `return_sequences` parametresini sonuncu hücreye gelene kadar `True` olarak ayarlamak zorundayız, böylece hücrelerin işlemleri tamamlanınca, içindeki veriler bir sonraki hücreye aktarılabilecek.
- Son `LSTM` hücresini eklerken herhangi bir return işlemi yapmasına gerek yoktur. Bu zaten son hücre olduğu için sadece çıkan sonucu almak yeterli olacaktır.
- Son adım bittikten sonra, bir tane `Dense` katmanı eklenir. Bu katmanın giriş ve çıkışları birbirine bağlı ve eşit sayıdadır. Kendinden önceki tüm hücreler ile bağlantısı vardır.
- Modele tüm hücreler eklendikten sonra, model compile edilir. Bunu yaparken `optimizer` olarak `adam` metodunu kullanmaya karar verilmiştir. İlgili [3] kaynakta da belirtildiği gibi, hafıza kullanımı düşük ve performansı çok yüksek olan bu optimizer, en çok tercih edilen optimizer'ların başında gelmektedir.
- Compile işlemi tamamlandıktan sonra, önce train verisi kullanılarak modelin `fit` fonksiyonu ile train edilmesi gerekmektedir. Burada modelin eğitimi sırasında, tüm veri aynı anda modele gönderilmemektedir. Onun yerine, bir parçası gönderilip, gelen sonuca göre ağırlıklar güncellenip tekrar gönderilmektedir. Bu her bir adıma `EPOCH` denilmektedir. Rapor kısmında `EPOCH` değerinin, model üzerindeki etkisini görebilirsiniz.
- Predict işlemi ardından çıkan sonuçları, `inverse_transform` fonksiyonu ile gerçek değerlerine çevirmemiz gerekmektedir. Çünkü üstteki adımlarda bu sıcaklık değerlerini 0 ile 1 arasındaki değerlere eşitlemiştik.
- Elimizde artık tahmini sıcaklık değerleri (modelden ürettiğimiz) ve gerçekleşen (CSV içinde) sıcaklık değerleri mevcuttur. Tek yapmamız gereken bu iki değeri görselleştirmektir. Bunun için matplotlib kütüphanesini kullanarak ilgili grafikleri çizmemiz gerekmektedir.
- Kodun en sonunda, elde ettiğimiz grafikleri bulunduğumuz dizine kaydedip, ilgili çıktıları bir metin belgesine yazıyoruz. Bu belge içerisinde `EPOCHS`, `BATCH SIZE`, `TIMESTAMP`, `Mean Squared Error`, $R^2$ değerlerini görebiliyoruz.

## VII. Tahmin Sonuçları

### A. Parametreler

İlgili testler sırasında birçok parametre ve model test etmiş bulunmaktayım. Bunlardan bazıları direkt olarak veriye, bazıları ise algoritmanın kendisine müdahale eden parametrelerdi. Araştırmalarım sonucunda bazı parametrelerin optimal değerlerini sabitleyip, ciddi fark yaratacak parametreleri değiştirip sonuçları kıyasladım.

| Değişken | Değer |
|---|---|
| `BATCH SIZE` | 8 |
| `TIMESTAMP` | 48 |
| `Gaussian Filter Sigma` | 1.2 |
| `Train-Test Size` | %80 - %20 |
| `LSTM Units` | 100 |

Tablo III
SABIT TUTULAN PARAMETRELER.

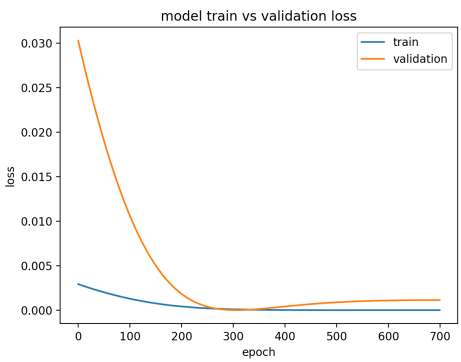
### B. Tahminler

Testlerimi ağırlıklı olarak `EPOCHS` değişkeni üzerinde yaptım. Aşağıdaki tabloda `EPOCHS` değerlerinin sonuçlar üzerindeki etkisini görebilirsiniz.

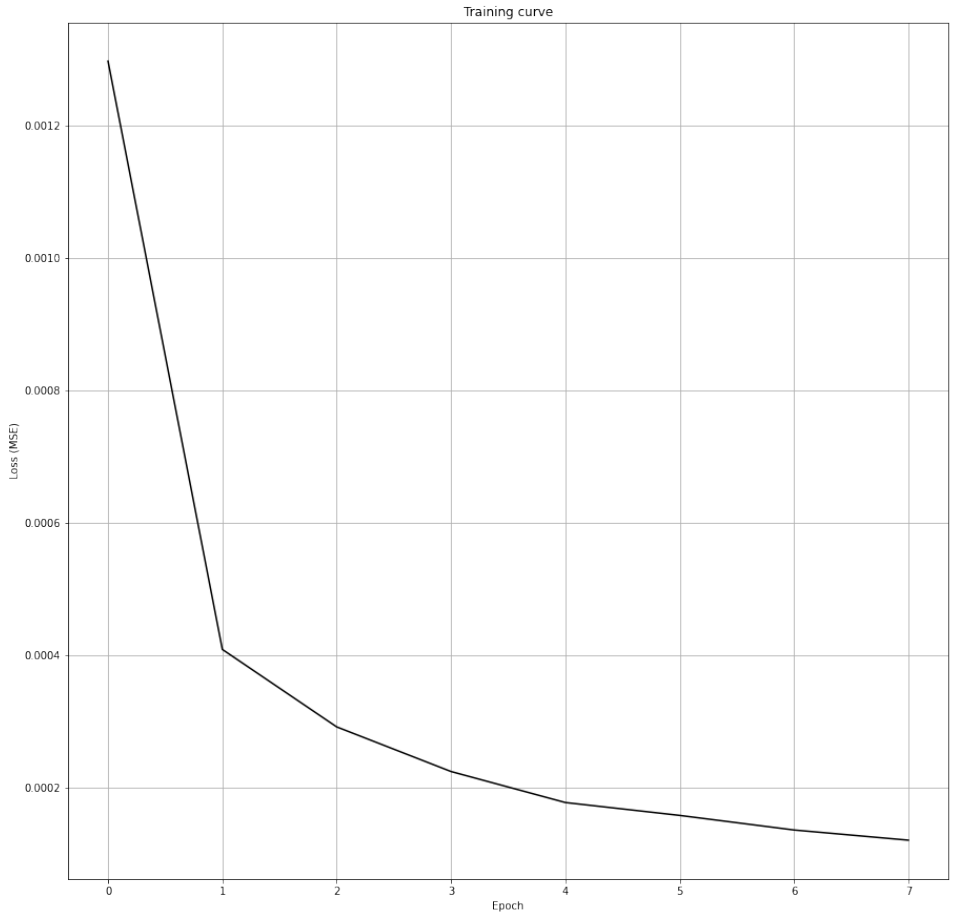
| Değer | Mean Square Error | $R^2$ |
|---|---|---|
| 1 | 1.78 | 0.88 |
| 2 | 0.73 | 0.89 |
| 3 | 0.67 | 0.92 |
| 4 | 0.55 | 0.93 |
| 5 | 0.47 | 0.95 |
| 6 | 0.40 | 0.96 |
| 7 | 0.36 | 0.97 |
| **8** | **0.23** | **0.98** |
| 9 | 0.39 | 0.96 |
| 10 | 0.75 | 0.92 |

Tablo IV
EPOCH DEĞERININ, TAHMIN SONUÇLARI ÜZERINDEKI ETKISI

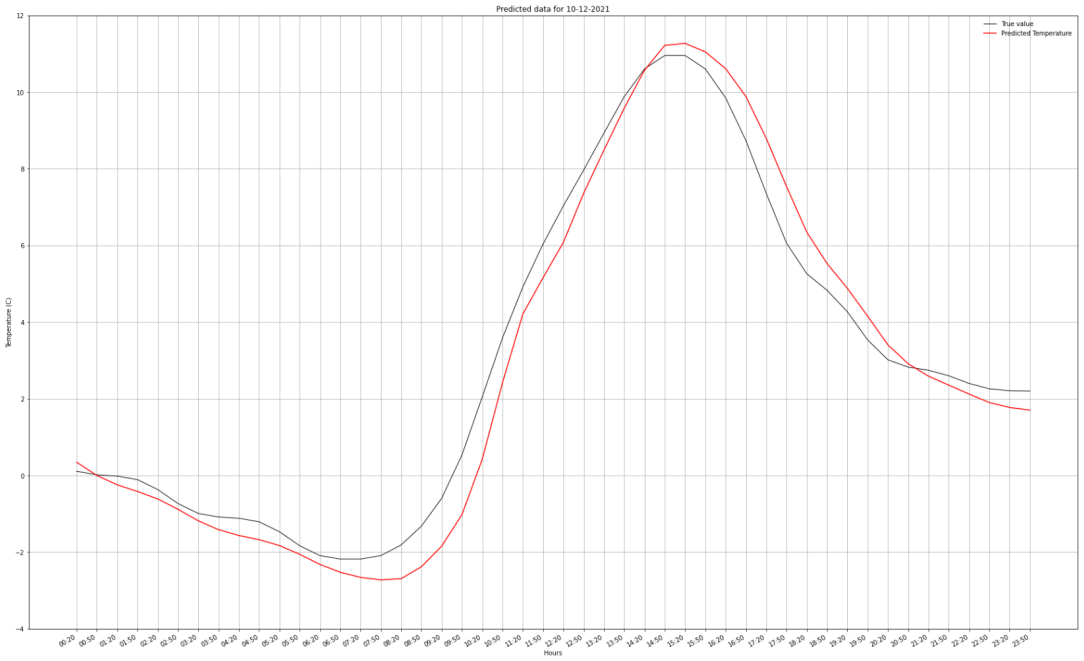
Tablo IV'de görüldüğü gibi, `EPOCHS` değerini bir noktadan sonra ne kadar arttırırsak o kadar çok gerçek değerden uzaklaşıyor. Buna overfit durumu deniliyor.




Şekil 4. Yüksek EPOCH değerlerinde meydaha gelen overfitting durumu.




Şekil 5. EPOCH değeri 8 olan iterason için MSE(Mean Squared Error) grafiği




Şekil 6. EPOCH değeri 8 olan iterason ile üretilen tahmin ve gerçekleşen veri grafiği

Tablo IV görebileceğimiz gibi, `EPOCH` değeri 8 olan iterasyon, çok iyi bir başarı göstermiştir.

Şekil 6 incelendiğinde, tahmin verilerinin oluşturduğu çizginin başarılı bir şekilde gerçek veriler ile örtüştüğünü görebiliriz. Bazı değerleri neredeyse %100 doğruluk payı ile tahmin ederken, bazı noktalarda ise gerçek veriden uzaklaştığını görebiliriz.

## VIII. Sonuç

Bu dökümanda LSTM algoritmasını, belirli bir veri seti ile beraber kullanarak gelecek için hava durumu tahmini üretmeye çalıştım. Elde ettiğim sonuçlar gayet başarılıydı. LSTM modelinin daha iyi sonuçlar verdiğini gördüğüm başka veri setleri ile test edilmiş örnekleri de inceledim.

Modelin sınırlarını zorlamak için `EPOCHS`, `BATCH SIZE` veya `TIMESTAMP` parametrelerini değiştirip bazı testler yaptım fakat veri setinin büyüklüğü sebebiyle süre ve güç tüketiminde inanılmaz artışlar yaşadım.

Elimdeki veri ile 1 EPOCH iterasyonunun tamamlanması yaklaşık **4 dakika** sürüyor. Şekil 4'deki gibi çok iyi MSE değeri elde etmek için 300 EPOCH iterasyonu yapmak istesem, bu **20 saat** gibi bir zaman alacaktır. Ayrıca elimdeki veri, birbirine çok yakın değerlerden ve azda olsa belli bir pattern ile gittiğinden, yüksek EPOCH değerleri bana yarardan çok zarar verecektir.

Yüksek EPOCH değeri ile, çok daha kompleks bir veri setini nasıl optimize ederim diye bir araştırma yaptım. LSTM algoritmasının diğer RNN algoritmalarına kıyasla çok daha fazla işlem gücüne ihtiyaç duyduğunu, bu durumu kontrol altına almak için NVIDIA CUDA teknolojisini kullanmak gerektiğini öğrendim. Gene NVIDIA tarafından geliştirilen cuDNN, 10x hız artışı sağlayarak RNN algoritmalarının sınırları zorlamasına yardımcı olduğunu öğrendim [4].

Kaynak koduma [5] aracılığı ile erişebilirsiniz.

## Kaynaklar


[1] Ankara Weather History. Weather Underground. (n.d.). Retrieved from https://www.wunderground.com/history/daily/tr/ankara/LTAC

[2] Hochreier, S., & Schimidhüber, J. (1997). Long Short Term Memory. Retrieved from https://www.bioinf.jku.at/publications/older/2604.pdf

[3] Kingma, D. P., & Ba, J. (2017, January 30). Adam: A method for stochastic optimization. arXiv.org. Retrieved from https://arxiv.org/abs/1412.6980

[4] Appleyard, J. (2016, April 6). Optimizing recurrent neural networks in cudnn 5. NVIDIA. Retrieved from https://developer.nvidia.com/blog/optimizing-recurrent-neural-networks-cudnn-5/

[5] https://github.com/ozankoyuk/LSTM_weather_forecasting