**Laporan Tugas Pengenalan Pola**

Annisa Muzdalifa

23519011

**Abstract.** Arsitektur RNN dan LSTM banyak digunakan untuk task-task *machine learning* yang berkaitan dengan keterurutan seperti POS-Tagging pada kalimat, atau fitur rekomendasi belanja. Tugas kali ini mengimplementasikan RNN dan LSTM kemudian mengaplikasikan arsitektur tersebut dalam beberapa eksperimen. Eksperimen mengubah beberapa parameter dalam arsitektur RNN dan LSTM untuk menghasilkan error yang minimum.

**1 Pendahuluan**

Pembelajaran mesin adalah salah satu aspek yang berhubungan dengan pengenalan pola. Lebih lanjut lagi penggunaan *deep learning* untuk memprediksi dari data sudah banyak dikembangkan. Salah satu arsitektur yang banyak digunakan yaitu *recurrent neural network*s (RNN). RNN juga dikembangkan lebih lanjut menjadi *long short-term memory* (LSTM). Penggunaan kedua arsitektur didasarkan dari keterbatasan *neural network* biasa yang tidak memanfaatkan keterurutan data. Pada RNN perhitungan data ke-(i-1) akan digunakan untuk data ke-i sehingga penggunaannya banyak ditemukan untuk data yang memperhatikan urutan, misalnya *time-series*.

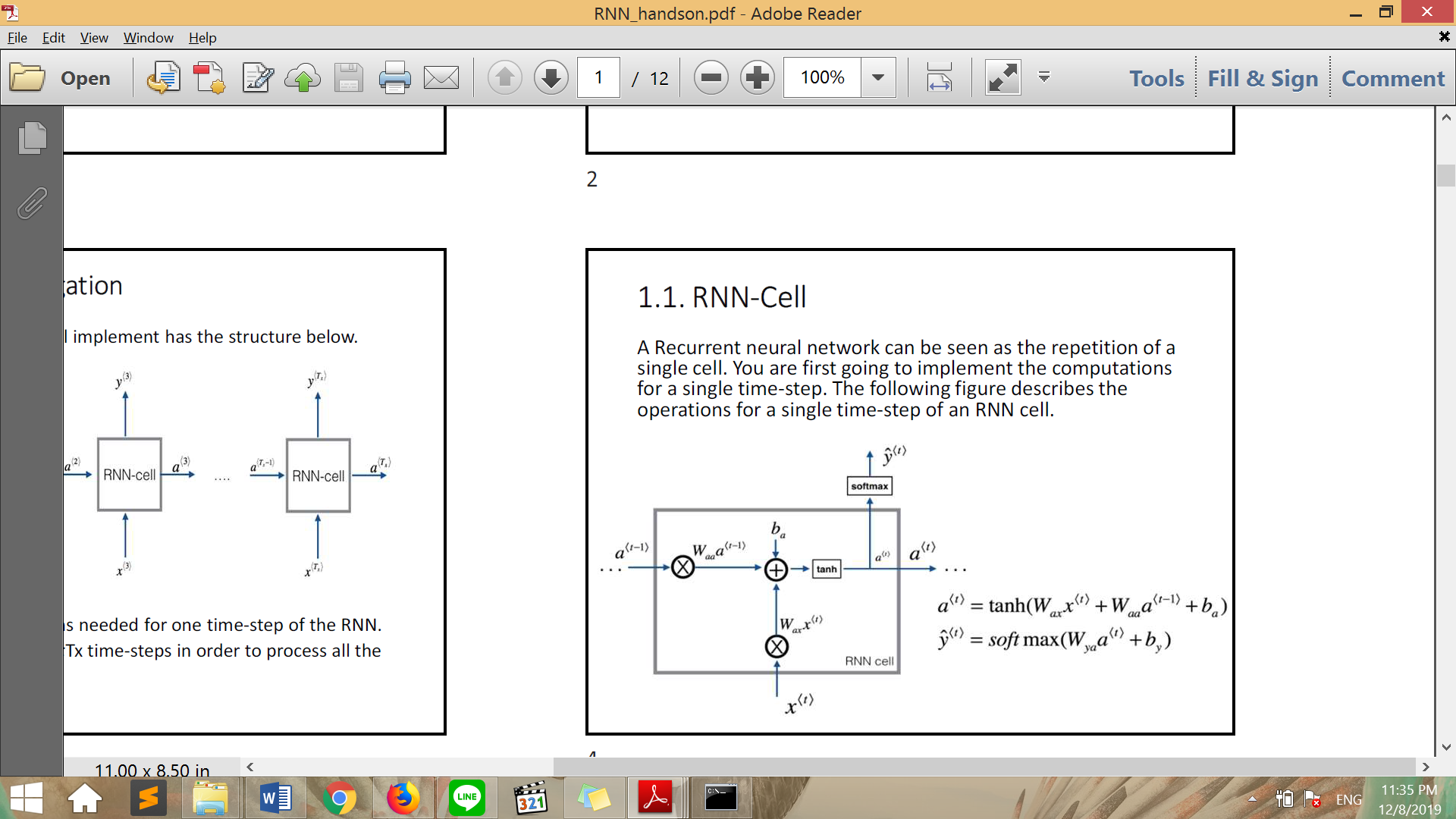
Pada tugas kali ini, diimplementasikan kakas untuk membangun arsitektur RNN dan LSTM serta eksperimen menggunakan tiga jenis data. Eksperimen pertama dilakukan untuk memprediksi harga saham dari dataset TATA dengan memanfaatkan kakas keras. Eksperimen kedua yaitu prediksi harga saham dari Apple dan masih memanfaatkan kakas keras. Eksperimen ketiga memanfaatkan *library* utils untuk memodelkan bahasa di level karakter dan mengeluarkan hasil akhir berupa nama dinosaurus. Laporan disusun sebagai berikut: Bab 2 menjelaskan implementasi RNN dan LSTM, Bab 3, 4, dan 5, memaparkan eksperimen secara berurutan, dan bab terakhir yaitu kesimpulan.

**2 RNN dan LSTM**

Pada Bab ini akan dijelaskan implementasi RNN dan LSTM yang dikerjakan. Subbab 2.1 menjelaskan implementasi RNN dan 2.2 menjelaskan implementasi LSTM.

**2.1 *Recurrent Neural Network* (RNN)**

RNN terdiri atas sel yang berulang. Dalam satu sel terdapat beberapa nilai yang diperhitungkan yaitu nilai dari input data dan nilai *hidden state* dari sel sebelumnya. Perhitungan dalam satu sel dapat dilihat pada Gambar 1.



Gambar 1. Perhitungan dalam satu sel RNN

Proses *training* pada RNN dapat dibagi menjadi dua bagian yaitu *forward propagation* dan *backward propagation*. *Forward propagation* adalah proses menghitung output yang telah dilewatkan ke dalam RNN. *Backward propagation* adalah proses untuk meperbarui weight agar *training* dapat mencapai konvergen. Implementasi RNN dibagi menjadi beberapa fungsi dalam Tabel 1.

**Tabel 1**. Fungsi pada RNN

|  |  |  |  |
| --- | --- | --- | --- |
| No | Nama | Parameter | Keterangan |
| 1 | rnn\_cell\_forward | xt: input  a\_prev: nilai *hidden* *state* sel sebelumnya  parameters: nilai weight dan bias | Melakukan *forward propagation* dalam satu sel |
| 2 | rnn\_forward | x: input  a0: nilai awal *hidden* *state*  parameters: nilai *weight* dan bias | Melakukan *forward propagation* lengkap. Memangil rnn\_cell\_forward sebanyak n *time-step* |
| 3 | rnn\_cell\_backward | da\_next: delta *weight* sel setelahnya  cache: hasil *forward propagation* | Menghitung *delta weight* dari satu sel |
| 4 | rnn\_backward | da: *delta weight*  caches: hasil *forward propagation* | Menghitung akumulasi *delta* *weight* dari satu kali propagasi. Memanggil rnn\_cell\_backward sebanyak n *time-step* |

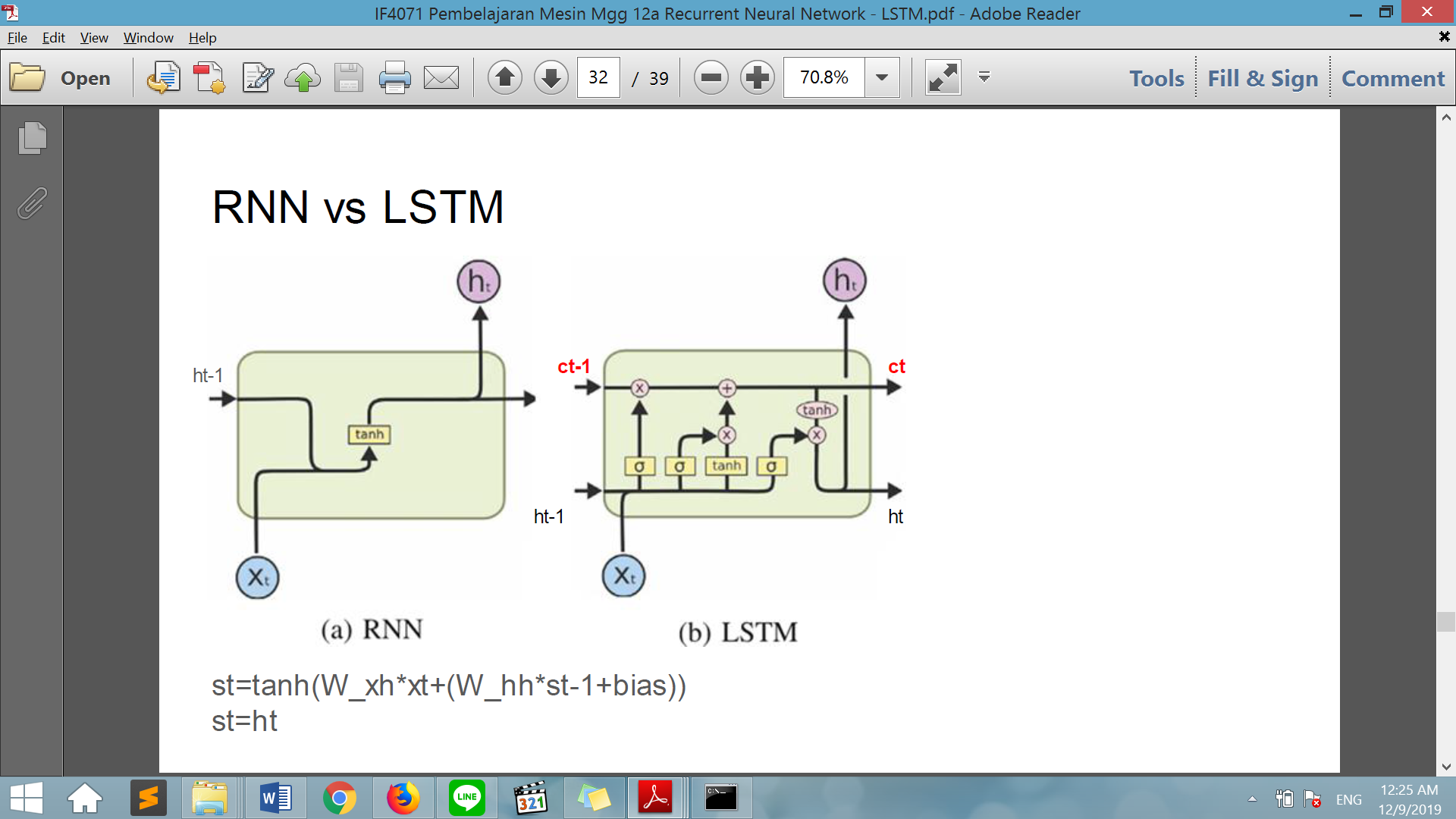
Berikut contoh satu kali propagasi melalui sel RNN dengan memanggil fungsi-fungsi tersebut.

|  |
| --- |
| # Inisialisasi random data  np.random.seed(1)  x = np.random.randn(3,10, 4)  a0 = np.random.randn(5, 10)  Waa = np.random.randn(5,5)  Wax = np.random.randn(5,3)  Wya = np.random.randn(2,5)  ba = np.random.randn(5,1)  by = np.random.randn(2,1)  parameters = {"Waa": Waa, "Wax": Wax, "Wya": Wya, "ba": ba, "by":by}  # Satu kali propagasi maju  a, y, caches = rnn\_forward(x, a0, parameters)  da = np.random.randn(5, 10, 4)  # Satu kali propagasi mundur  gradients = rnn\_backward(da, caches)  # Perbedaan Weight setelah backpropagation  print(parameters['Waa'])  print(gradients['dWaa']) |
| [[-0.64691669 0.90148689 2.52832571 -0.24863478 0.04366899]  [-0.22631424 1.33145711 -0.28730786 0.68006984 -0.3198016 ]  [-1.27255876 0.31354772 0.50318481 1.29322588 -0.11044703]  [-0.61736206 0.5627611 0.24073709 0.28066508 -0.0731127 ]  [ 1.16033857 0.36949272 1.90465871 1.1110567 0.6590498 ]]  [[ 0.5790693 1.00989782 0.45607653 0.47247531 0.42070208]  [-4.0169478 1.27988543 0.44183867 -3.005149 3.68367168]  [-1.58623708 -1.82523051 1.88662745 1.25372645 1.31696213]  [-4.06220832 -2.19095447 -2.34204406 -2.90812842 3.40599372]  [-1.52082597 -3.22930103 -3.15030367 2.82807664 0.79115841]] |

Pada hasil implementasi di atas, dapat dilihat bahwa nilai weight dari hidden state ke hidden state lainnya berubah. Terdapat 25 nilai weight, 5 untuk masing-masing *hidden state* yang memiliki 5 unit. Nilai *weight* yang sesuai akan menghasilkan konvergen untuk *data train*.

**2.2 *Long Short-Term Memory* (LSTM)**

LSTM adalah salah satu perbaikan dari RNN yang mengatasi masalah vanishing gradient. Arsitektur besar LSTM masih sama dengan RNN yaitu sel yang berulang-ulang. Perbedaan terletak dalam perhitungan di dalam sel. LSTM memiliki beberapa gate yang digambarkan dalam Gambar 2.



Gambar 2. Arsitektur satu sel LSTM

Perhitungan dalam sel LSTM dapat dibagi menjadi cell state, forget gate layer, input gate layer, dan output layer. Pada Tabel 2. ditampilkan fungsi yang diimplementasi dalam LSTM.

**Tabel 2.** Fungsi pada LSTM

|  |  |  |  |
| --- | --- | --- | --- |
| No | Nama | Parameter | Keterangan |
| 1 | lstm\_cell\_forward | xt: input  a\_prev: nilai *hidden* *state* sel sebelumnya  c\_prev: nilai *memory state* sel sebelumnya  parameters: nilai *weight* dan bias | Melakukan *forward propagation* dalam satu sel |
| 2 | lstm\_forward | x: input  a0: nilai awal *hidden* *state*  parameters: nilai *weight* dan bias | Melakukan *forward propagation* lengkap. Memangil lstm\_cell\_forward sebanyak n *time-step* |
| 3 | lstm\_cell\_backward | da\_next: delta *weight* *hidden state* sel setelahnya  dc\_next: delta *weight* *memory state* sel setelahnya  cache: hasil *forward propagation*  dbo: *gradient* bias output | Menghitung *delta weight* dari satu sel |

Berikut contoh satu kali propagasi dalam satu sel menggunakan fungsi yang telah diimplementasi di atas.

|  |
| --- |
| # Inisialisasi data secara random  np.random.seed(1)  xt = np.random.randn(3, 10)  a\_prev = np.random.randn(5, 10)  c\_prev = np.random.randn(5, 10)  Wf = np.random.randn(5, 5+3)  bf = np.random.randn(5, 1)  Wi = np.random.randn(5, 5+3)  bi = np.random.randn(5, 1)  Wo = np.random.randn(5, 5+3)  bo = np.random.randn(5, 1)  Wc = np.random.randn(5, 5+3)  bc = np.random.randn(5, 1)  Wy = np.random.randn(2, 5)  by = np.random.randn(2, 1)  parameters = {"Wf": Wf, "bf": bf, "Wi": Wi, "bi":bi, "Wo":Wo, "bo":bo, "Wc":Wc, "bc":bc, "Wy":Wy, "by":by}  # propagasi maju dalam satu sel LSTM  a\_next, c\_next, yt, cache = lstm\_cell\_forward(xt, a\_prev, c\_prev, parameters)  da\_next = np.random.randn(5, 10)  dc\_next = np.random.randn(5, 10)  # update weight dengan propagasi mundur dalam satu sel LSTM  gradients = lstm\_cell\_backward(da\_next, dc\_next, cache)  print(Wf)  print(gradients['dWf']) |
| [[-0.1809203 -0.60392063 -1.23005814 0.5505375 0.79280687 -0.62353073  0.52057634 -1.14434139]  [ 0.80186103 0.0465673 -0.18656977 -0.10174587 0.86888616 0.75041164  0.52946532 0.13770121]  [ 0.07782113 0.61838026 0.23249456 0.68255141 -0.31011677 -2.43483776  1.0388246 2.18697965]  [ 0.44136444 -0.10015523 -0.13644474 -0.11905419 0.01740941 -1.12201873  -0.51709446 -0.99702683]  [ 0.24879916 -0.29664115 0.49521132 -0.17470316 0.98633519 0.2135339  2.19069973 -1.89636092]]  [[-0.1421699 0.37848152 0.11926438 0.47170657 0.28198475 0.03379451  -0.14190739 0.11263537]  [ 0.33483087 0.28531747 0.19488761 0.49997521 0.71242185 -0.43398358  -0.65500412 0.44600686]  [-0.06376814 -0.06822228 -0.00809406 0.01610937 -0.05571127 0.14833691  0.10106896 0.04564574]  [ 0.0621335 -0.14795484 0.12115521 -0.7750645 0.0157104 -0.20879551  -0.02786287 0.39423542]  [ 0.05048145 -0.49088288 0.4095815 0.03122771 -0.33133813 0.88204865  0.55665687 0.12335136]] |

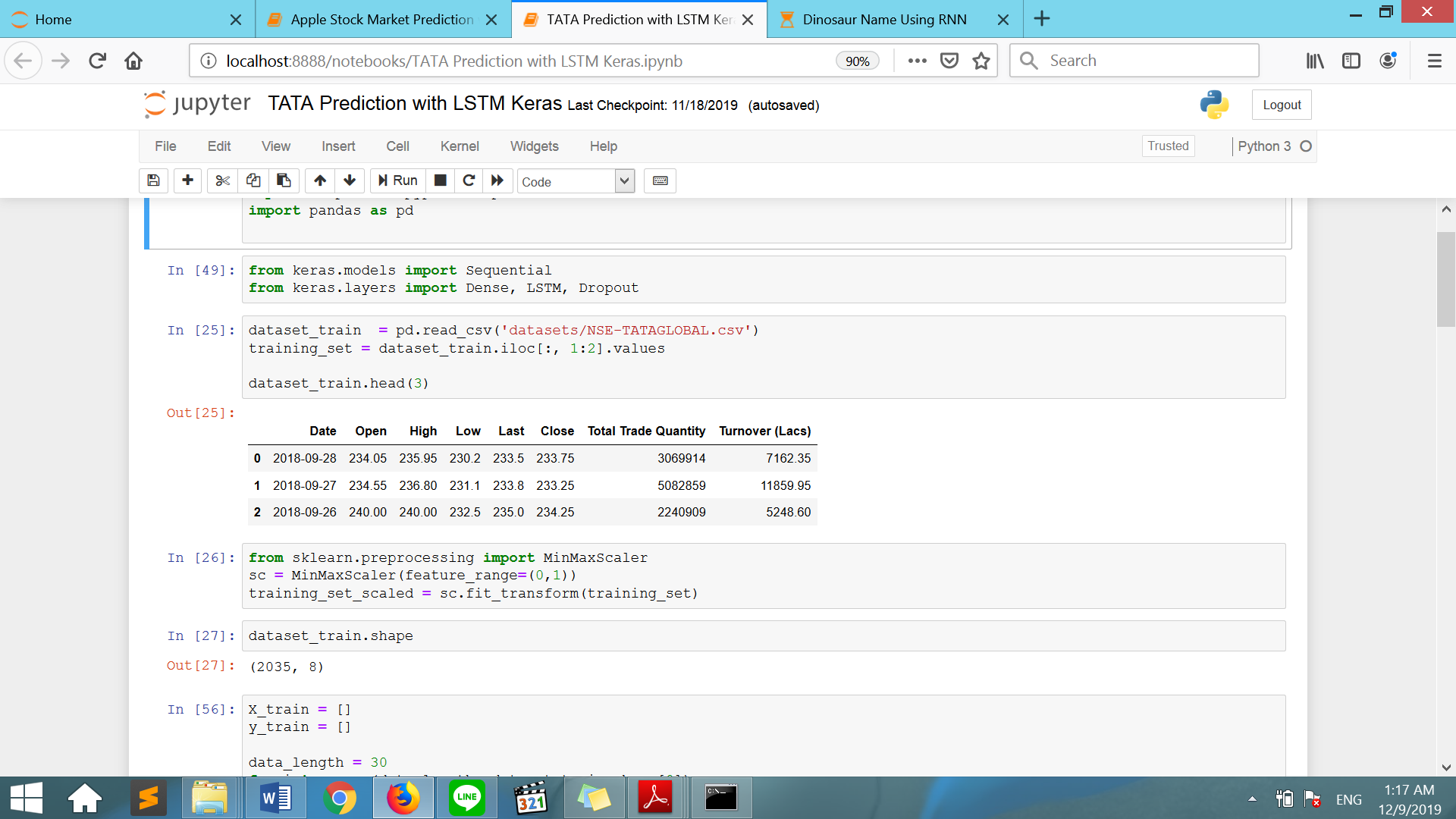
Dalam hasil yang ditampilkan di atas bobot dari forget gate berubah setelah fungsi *backward* dipanggil. Bobot terbaik diperoleh ketika sudah konvergen.

**3 Stock Market Prediction dengan LSTM**

Pada Bab ini akan dijelaskan eksperimen menggunakan LSTM terhadap dataset time-series. Eksperimen bertujuan untuk memprediksi harga saham berdasarkan harga di hari-hari sebelumnya. Jumlah dataset keseluruhan adalah 2035 dan *task* yang dilakukan adalah regresi. Bab ini dibagi menjadi beberapa subbab yaitu preproses, eksperimen, dan hasil.

**3.1 Preproses**

Sebelum data dimasukkan ke dalam proses training, data harus di proses karena agar data sesuai dengan requirement dan memberi metric score yang lebih baik. Pada eksperimen ini digunakan dataset TATA dengan struktur sebagai berikut.



Nilai yang akan di prediksi adalah kolom Open yaitu menunjukkan harga saham sehingga data lain dapat diabaikan. Dataset dipotong hingga terdapat satu kolom saja yaitu sebagai berikut.

|  |
| --- |
| [[234.05]  [234.55]  [240. ]  [233.3 ]  [233.55]  [235. ]  …  [117.6]  [120.1]  [121.8]  [120.3]  [122.1]] |

Berdasarkan dataset di atas, nilai yang muncul sangat bervariatif. Pada machine learning, data yang di proses adalah angka sehingga lebih baik jika nilai tersebut ditransformasi menjadi skala 0-1.

|  |
| --- |
| [[0.6202352]  …  [0.14801298]  [0.15815085]  [0.16504461]  [0.15896188]  [0.16626115]] |

Karena LSTM menerima input dalam bentuk time-series, maka perlu ditetapkan panjang dari satu time-step. Pada eksperimen kali ini, diambil panjang satu time-step adalah 30. Oleh karena itu, data perlu diubah agar memenuhi dimensi input yang dimiliki LSTM. Satu instance akan terdiri atas 30 data harga saham sebelumnya. Berikut instance ke-0 dari yang asalnya hanya satu dimensi yaitu 0.6202352.

|  |
| --- |
| [[0.620235198702352], [0.6222627737226277], [0.6443633414436334], [0.6171938361719383], [0.6182076236820763], [0.6240875912408759], [0.6279399837793997], [0.6358475263584753], [0.6165855636658556], [0.577250608272506], [0.5484590429845904], [0.5733982157339821], [0.5733982157339821], [0.5673154906731549], [0.5794809407948094], [0.5713706407137064], [0.6370640713706407], [0.6281427412814273], [0.6321978913219789], [0.6255068937550688], [0.6194241686942417], [0.6321978913219789], [0.6111111111111112], [0.6220600162206001], [0.6455798864557989], [0.6723438767234386], [0.6605839416058394], [0.6476074614760746], [0.628345498783455], [0.6240875912408759]] |

Setelah dataset dibentuk ke dalam time-series, barulah data tersebut dapat dimasukkan kedalam arsitektur LSTM

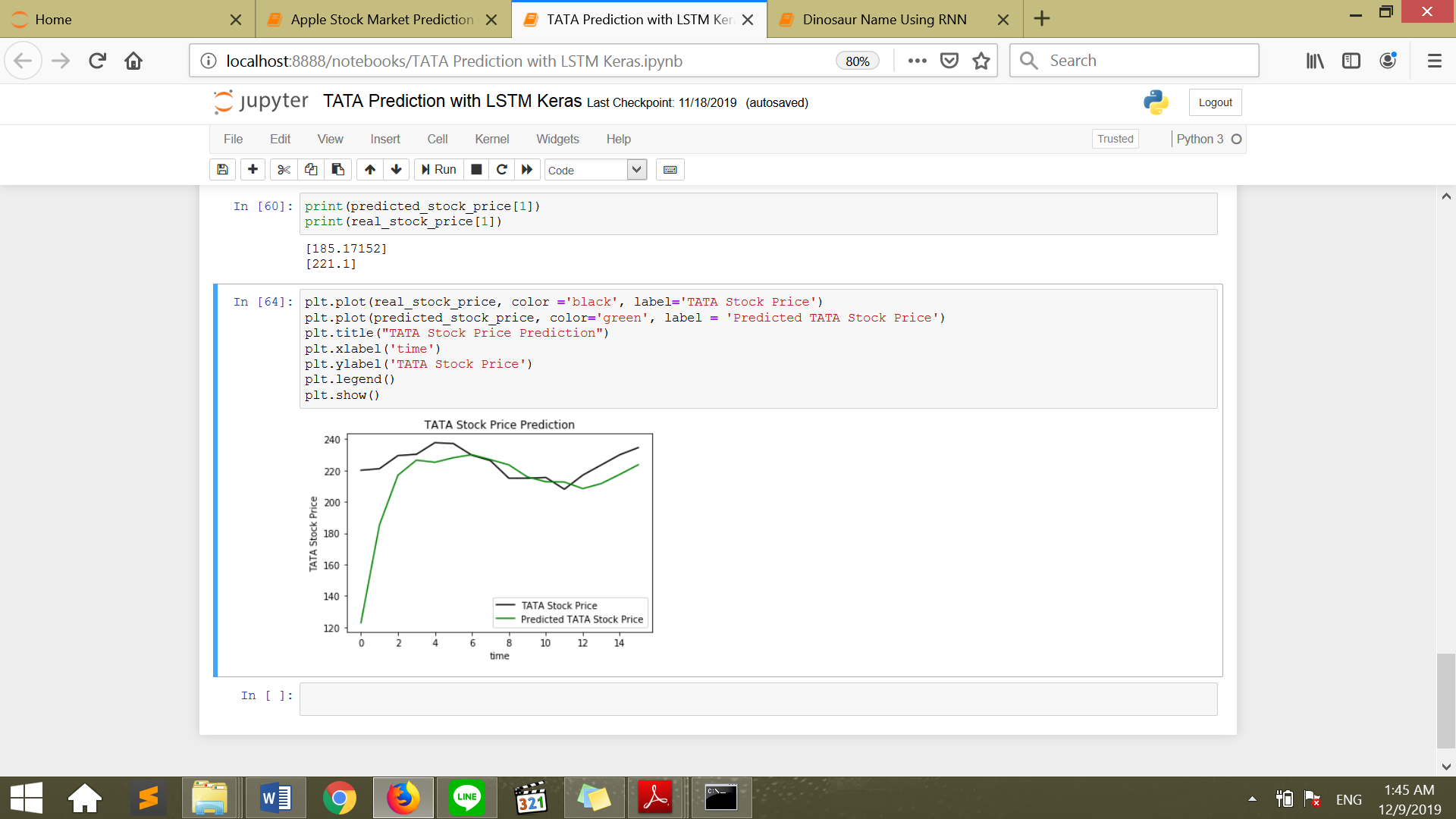
**3.2 Eksperimen**

Eksperimen ini tidak mengubah arsitektur yang terdiri atas 4 layer LSTM beserta *dropout* dan satu *dense layer* dengan satu neuron output. Optimizer yang digunakan adalah adam dan metric yang dipilih yaitu *mean squared error*. Eksperimen dilakukan dengan mengubah beberapa hal diantaranya panjang time-step, jumlah epoch, dan jumlah batch size. Berikut hasil eksperimen disajikan dalam Tabel 3.

**Tabel 3.** Training Dataset TATA

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | N input | Epoch | Batch size | Loss |
| Baseline | 60 | 100 | 32 | 7.63 x 10-4 |
| 1 | 60 | 200 | 32 | 6.29 x 10-4 |
| 2 | 60 | 100 | 16 | 6.04 x 10-4 |
| 3 | **30** | **100** | **32** | **5.54 x 10-4** |
| 4 | 30 | 200 | 32 | 5.86 x 10-4 |
| 5 | 30 | 100 | 16 | 6.29 x 10-4 |

Hasil prediksi terhadap data test ditampilkan dalam Gambar 3. Data test melalui preproses yang sama dengan data train. Untuk menampilkan prediksi terhadap harga sebenernya, maka data ditransform kembali menjadi skala semula dengan melakukan inverse.



Gambar 3. Hasil Prediksi LSTM terhadap Dataset TATA

**3.3 Analisis**

Jumlahn-input menunjukkan panjang satu time-step yang akan dimasukkan ke dalam LSTM. Jumlah ini dapat disesuaikan tergantung domain dari data, n-input 30 untuk stock market prediction dapat diartikan bahwa data hari ini dilihat kurang lebih dari data selama satu bulan kebelakang. N-input sebesar 30 memberikan error lebih kecil dibandingkan 60 bisa jadi karena data satu bulan lebih relevan dibanding dua bulan sebelumnya. Jumlah epoch menunjukkan banyaknya pengulangan terhadap satu data training. Semakin besar jumlah epoch semakin besar kemungkinan training mencapai konvergen. Namun, jumlah epoch 100 sudah mencapai konvergen karena errornya tidak berubah secara signifikan. Jumlah epoch tidak dinaikkan kembali untuk mengatasi terjadinya overfit. Batch size yang menghasilkan error minimum yaitu 32 untuk n- artinya bobot diperbarui setelah 32 kali propagasi maju dilakukan. Semakin kecil nilai batch size maka semakin sering bobot diperbarui sehingga diharapkan mencapai konvergen lebih cepat. Ketika batch size dikurang, error yang dihasilkan tidak berkurang banyak, sehingga ditetapkan menjadi 32 saja.

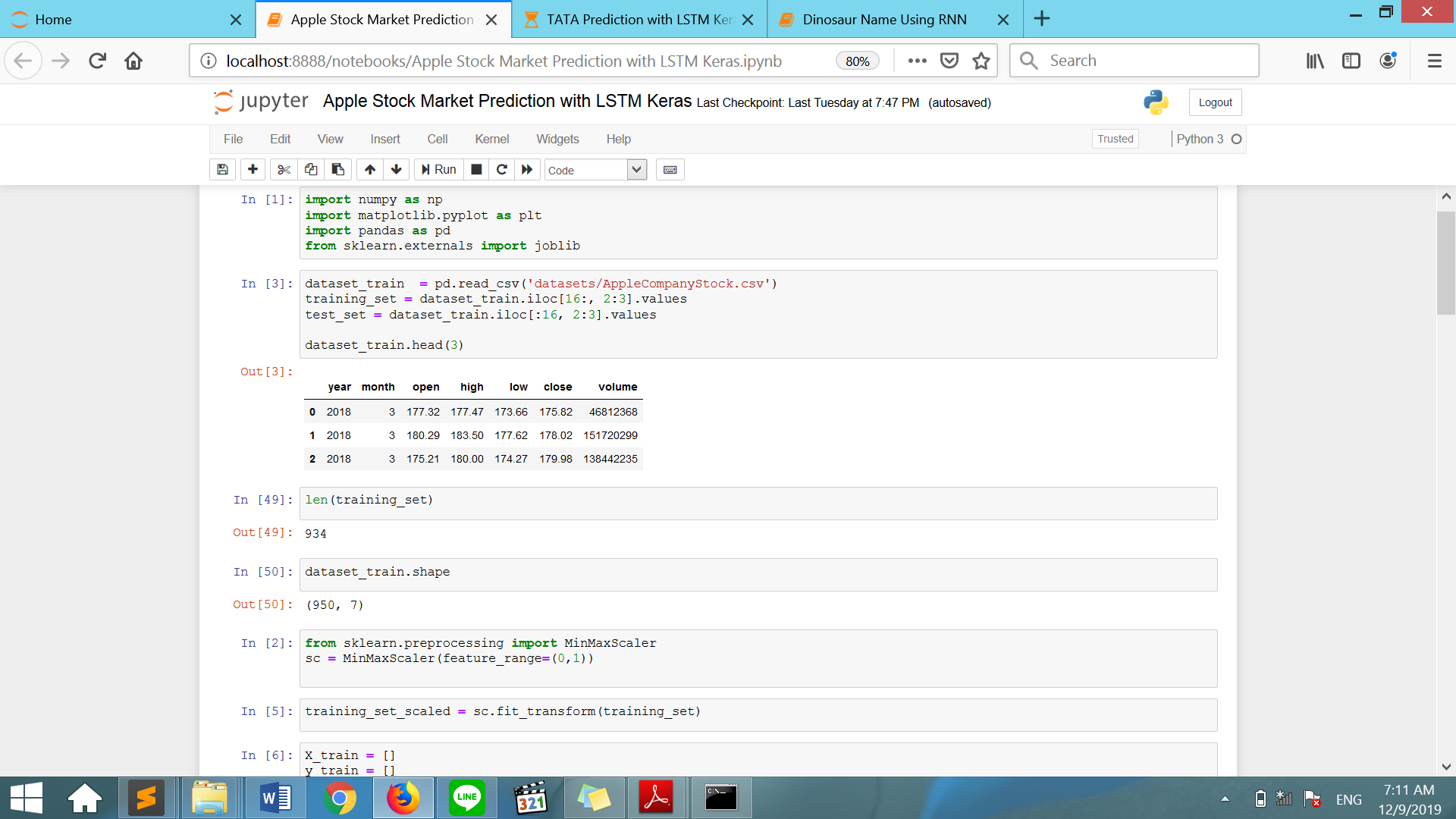
Dari hasil prediksi dalam Gambar 3, kinerja LSTM mendekati nilai sebenarnya, hanya saja pada data pertama perbedaannya sangat jauh. Selain itu, dilihat dari grafik, LSTM secara umum dapat memprediksi pola misalnya kenaikan dari data ke-12 sampai ke-14. Terdapat pula prediksi yang tidak sesuai misalnya penurunan harga berdasarkan prediksi dari data ke-8 sampai data ke-10 sedangkan data asli mengalami kenaikan.

**4 Apple Market Prediction dengan LSTM**

Pada Bab ini akan dijelaskan eksperimen menggunakan LSTM terhadap dataset time-series. Eksperimen bertujuan untuk memprediksi harga saham apple berdasarkan harga di hari-hari sebelumnya. Jumlah dataset keseluruhan adalah 950 dan *task* yang dilakukan adalah regresi. Bab ini dibagi menjadi beberapa subbab yaitu preproses, eksperimen, dan hasil.

**4.1 Preproses**

Sebelum data dimasukkan ke dalam proses training, data harus di proses karena agar data sesuai dengan requirement dan memberi metric score yang lebih baik. Pada eksperimen ini digunakan dataset Apple dengan struktur sebagai berikut.



Karena dataset tidak dipisahkan train dengan test, maka dilakukan split sehingga 16 data awal dijadikan untuk test, dan sisanya train. Nilai yang akan di prediksi adalah kolom open yaitu menunjukkan harga saham per minggu sehingga data lain dapat diabaikan. Dataset dipotong hingga terdapat satu kolom saja yaitu sebagai berikut.

|  |
| --- |
| [[175.05 ]  [170.29 ]  [173.5 ]  [172.365]  [163.89 ]  …  [108. ]  [101. ]  [108.44]  [101. ]  [102. ]] |

Sama halnya dengan eksperimen pada Bab 3, nilai yang muncul sangat bervariatif sehingga data tersebut ditransformasi menjadi skala 0-1.

|  |
| --- |
| [[0.23595931]  …  [0.13825012]  [0.12804931]  [0.13889132]  [0.12804931]  [0.12950657]] |

Karena LSTM menerima input dalam bentuk time-series, maka perlu ditetapkan panjang dari satu time-step. Pada eksperimen kali ini, diambil panjang satu time-step adalah 9 yang merempresentasikan harga saham dalam 2 bulan. Oleh karena itu, data perlu diubah agar memenuhi dimensi input yang dimiliki LSTM. Satu instance akan terdiri atas 9 data harga saham sebelumnya. Berikut instance ke-0 dari yang asalnya hanya satu dimensi yaitu 0.23595931.

|  |
| --- |
| [[0.2359593133397453], [0.22902276237941185], [0.23370056250182158], [0.23204657398501938], [0.21969630730669462], [0.2094954970709102], [0.2109673282620734], [0.2079216577773892], [0.20566290693946546]] |

Setelah dataset dibentuk ke dalam time-series, barulah data tersebut dapat dimasukkan kedalam arsitektur LSTM.

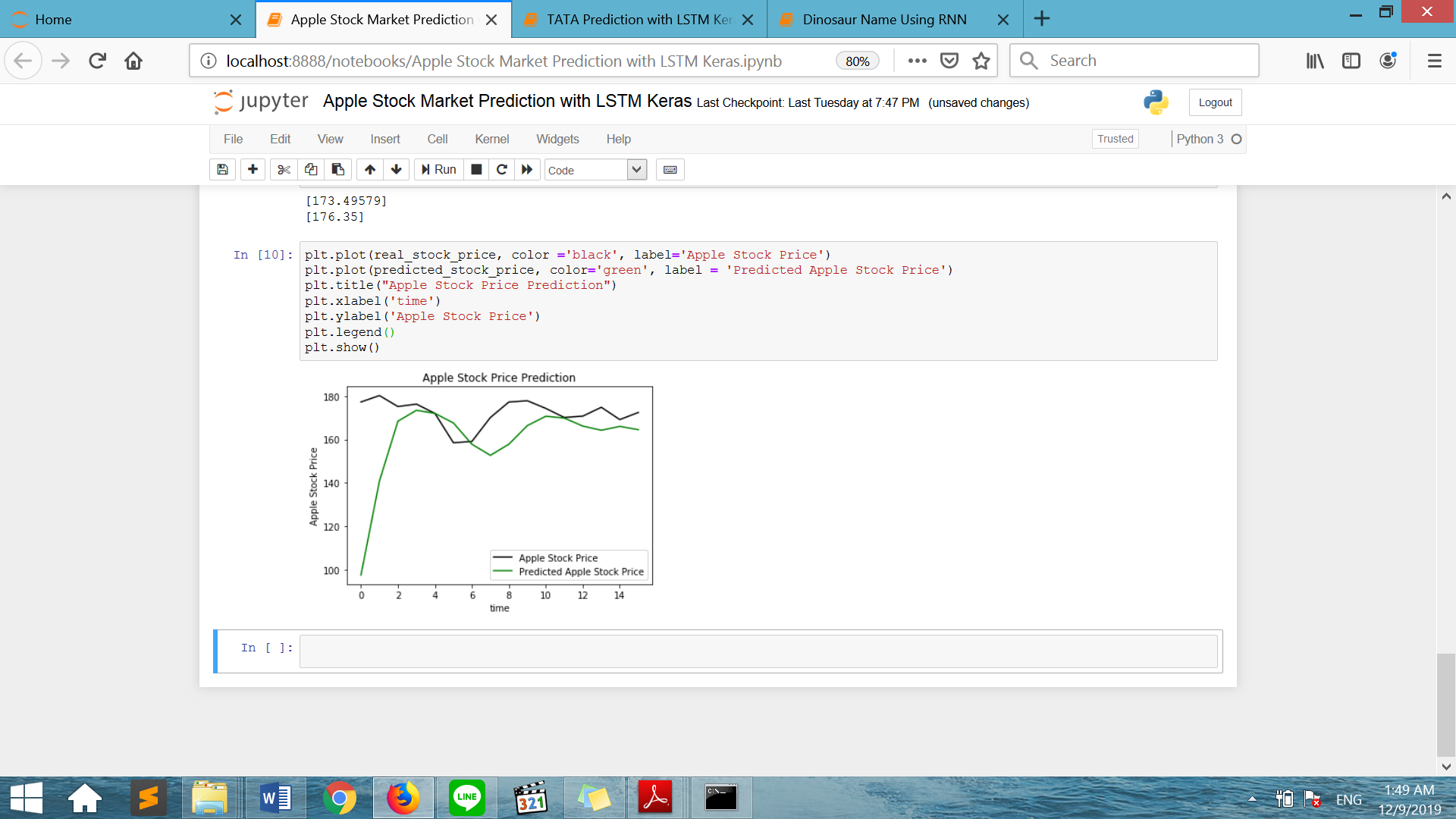
**4.2 Eksperimen**

Eksperimen ini tidak mengubah arsitektur yang terdiri atas 4 layer LSTM beserta *dropout* dan satu *dense layer* dengan satu neuron output. Optimizer yang digunakan adalah adam dan metric yang dipilih yaitu *mean squared error*. Eksperimen dilakukan dengan mengubah beberapa hal diantaranya panjang time-step, jumlah epoch, dan jumlah batch size. Berikut hasil eksperimen disajikan dalam Tabel 4.

**Tabel 4.** Training Dataset Apple

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | N input | Epoch | Batch size | Loss |
| Baseline | 9 | 100 | 32 | 0.0020 |
| 1 | 9 | 200 | 32 | 0.0019 |
| 2 | **9** | **100** | **16** | **0.0016** |
| 3 | 5 | 100 | 32 | 0.0021 |
| 4 | 5 | 200 | 32 | 0.0016 |
| 5 | 5 | 100 | 16 | 0.0016 |

Hasil prediksi terhadap data test ditampilkan dalam Gambar 4. Data test melalui preproses yang sama dengan data train. Jumlah data test yaitu 16 instance. Untuk menampilkan prediksi terhadap harga sebenernya, maka data ditransform kembali menjadi skala semula dengan melakukan inverse.



Gambar 4. Hasil Predikisi LSTM terhadap Dataset Apple

**4.3 Analisis**

Sama halnya dengan eksperimen pada Bab 3, jumlahn-input menunjukkan panjang satu time-step yang akan dimasukkan ke dalam LSTM. Jumlah ini dapat disesuaikan tergantung domain dari data, n-input 9 untuk dataset Apple dapat diartikan bahwa data hari ini dilihat kurang lebih dari data selama dua bulan kebelakang. Karena dataset Apple tidak melihat data perhari namun perminggu. N-input sebesar 9 memberikan error lebih kecil dibandingkan 5 bisa jadi karena data dua bulan lebih relevan dibanding hanya satu bulan sebelumnya. Selain itu, panjang input juga mempengaruhi dalam menangkap pola, jika berjumlah 5 saja maka terlalu sedikit. Jumlah epoch menunjukkan banyaknya pengulangan terhadap satu data training. Semakin besar jumlah epoch semakin besar kemungkinan training mencapai konvergen. Namun, jumlah epoch 100 sudah mencapai konvergen karena errornya tidak berubah secara signifikan. Jumlah epoch tidak dinaikkan kembali untuk mengatasi terjadinya overfit. Batch size yang menghasilkan error minimum yaitu 18. Semakin kecil nilai batch size maka semakin sering bobot diperbarui sehingga diharapkan mencapai konvergen lebih cepat. Ketika batch size berjumlah 32, error yang dihasilkan semakin besar disebabkan oleh data yang lebih sedikit sehingga semakin banyak update bobot semakin mendekati konvergen.

Dari hasil prediksi dalam Gambar 4, kinerja LSTM mendekati nilai sebenarnya, hanya saja data terlalu sedikit untuk diprediksi dengan menggunakan LSTM. Oleh sebab itu, prediksinya tidak terlalu baik dan errornya lebih besar dibandingkan dataset TATA. Dapat dilihat terdapat beberapa pola yang tidak sesuai dengan data sebenarnya, contohnya kenaikan pada dari data ke-14 ke data ke-15, diprediksi turun.

**5 Character Level Language Modeling**

Pada Bab ini akan dijelaskan eksperimen menggunakan RNN terhadap dataset dinosaurus. Dibagi menjadi beberapa subbab yaitu preproses, eksperimen, dan hasil.

**5.1 Preproses**

Sebelum menggunakan library utils untuk menghasilkan nama dinosaurus menggunakan RNN, perlu implementasi beberapa fungsi pada Tabel 5 berikut.

**Tabel 5.** Fungsi pada Character Level LM

|  |  |  |  |
| --- | --- | --- | --- |
| No | Nama | Parameter | Keterangan |
| 1 | clip | gradients: delta bobot  maxValue: nilai maksimum delta bobo | Melakukan clipping agar tidak terdapat gradient yang perbedaannya terlalu besar |
| 2 | sample | parameters: nilai *weight* dan bias  char\_to\_ix: mapping karakter kedalam integer  seed: penentu saat random | Melakukan sampling karakter dengan mengambil 50 karakter secara random |
| 3 | optimize | X: list data  Y: list target  a\_prev: nilai *hidden state* sel sebelumnya  parameters: nilai *weight* dan bias  learning\_rate: nilai learning rate | Menghitung satu kali propagasi maju, *update* bobot sesuai dengan *gradient* yang telah di *clip*. |
| 4 | model | data: Text corpus  ix\_to\_char: mapping index terhadap karakter  char\_to\_ix: mapping karakter kedalam integer  num\_iteration: jumlah epoch  n\_a: unit rnn  dino\_names: jumlah sampling  vocab\_size: jumlah karakter unik | Proses *training* untuk menghasilkan nama dinosaurus. |

Dari fungsi di atas, data yang dibaca adalah teks yang berisi list nama dinosaurus. Berikut teks korpus untuk training dengan jumlah 1536 nama

|  |
| --- |
| Aachenosaurus  Aardonyx  Abdallahsaurus  ... |

Dari data tersebut, diekstrak karakter apa saja yang muncul kemudian dipetakan ke dalam index sehingga menghasilkan map berikut

|  |
| --- |
| {0: '\n', 1: 'a', 2: 'b', 3: 'c', 4: 'd', 5: 'e', 6: 'f', 7: 'g', 8: 'h', 9: 'i', 10: 'j', 11: 'k', 12: 'l', 13: 'm', 14: 'n', 15: 'o', 16: 'p', 17: 'q', 18: 'r', 19: 's', 20: 't', 21: 'u', 22: 'v', 23: 'w', 24: 'x', 25: 'y', 26: 'z'} |

Setelah mapping tersedia, maka proses *training* dapat dilakukan dengan memanggil fungsi model yang telah diimplementasi.

**5.2 Eksperimen**

Eksperimen dilakukan dengan mencoba mengubah parameter iterasi dan jumlah sel dalam RNN. Berikut hasil eksperimen ditampilkan dalam Tabel 6.

**Tabel 6.** Training RNN

|  |  |  |  |
| --- | --- | --- | --- |
|  | Iterasi | Jumlah sel | Loss |
| 1 | **35000** | **50** | **22.318447** |
| 2 | 50000 | 50 | 22.409764 |
| 3 | 35000 | 100 | 23.684047 |

Dari eksperimen diatas, ditampilkan contoh nama yang digenerate oleh RNN dengan konfigurasi pertama yaitu jumlah iterasi 35000 dan jumlah sel 50 sebagai berikut

|  |
| --- |
| Iteration: 34000, Loss: 22.318447  Olusaurus  Klacaesaurus  Lusnsaurus  Ola  Ytosaurus  Egaisaurus  Trhangosaurus |

**5.3 Analisis**

Jumlah iterasi tidak menjamin error yang minimum, dari hasil eksperimen, ketika jumlah iterasi ditambah menjadi 50000 error tidak berkurang. Jumlah sel pada RNN juga tidak mengurangi error untuk iterasi yang sama. Nama dinosaurus yang dihasilkan tidak semuanya dapat dibaca, untuk hasil yang lebih maksimal diperlukan dataset yang lebih besar dan waktu komputasi yang lebih lama.

**6 Kesimpulan**

Tugas ini memiliki fokus pada eksplorasi arsitektur *neural network* dalam *machine learning* yaitu RNN dan LSTM. Berikut beberapa poin kesimpulan dari hasil pengerjaan tugas.

* Implementasi arsitektur RNN dan LSTM dengan bantuan library rnn\_utils mampu memberikan pemahaman lebih dalam terhadap kedua arsitektur. RNN dan LSTM memiliki perbedaan perhitungan di dalam satu selnya. Proses training pada keduanya terdiri atas propagasi maju dan propagasi mundur.
* Eksperimen LSTM terhadap dataset TATA menghasilkan loss sebesar 5.54x10-4. Selain jumlah layer pada LSTM, nilai epoch, batch size, serta panjang dari time-step, menentukan loss yang dihasilkan.
* Eksperimen LSTM terhadap dataset Apple menghasilkan loss sebesar 16x10-4. Hal ini bisa disebabkan karena jumlah dataset yang lebih sedikit dibandingkan dataset TATA
* Eksperimen RNN untuk membentuk nama dinosaurus baru menghasilkan error yang cukup besar. Memerlukan dataset yang lebih besar agar error lebih minimum. Jumlah iterasi, dan hidden layer dapat diubah untuk mencari error minimum.

**References**

1. Slide Kuliah IF5181 Pengenalan Pola 2019/2020
2. Slide Kuliah IF4071 Pembelajaran Mesin 2018/2019
3. Link tugas (<https://github.com/Amuzdalifa/IF5181-Tugas-23519011>)