Sprawozdanie

detekcja anomalii przy użyciu sieci rekurencyjnej- LSTM

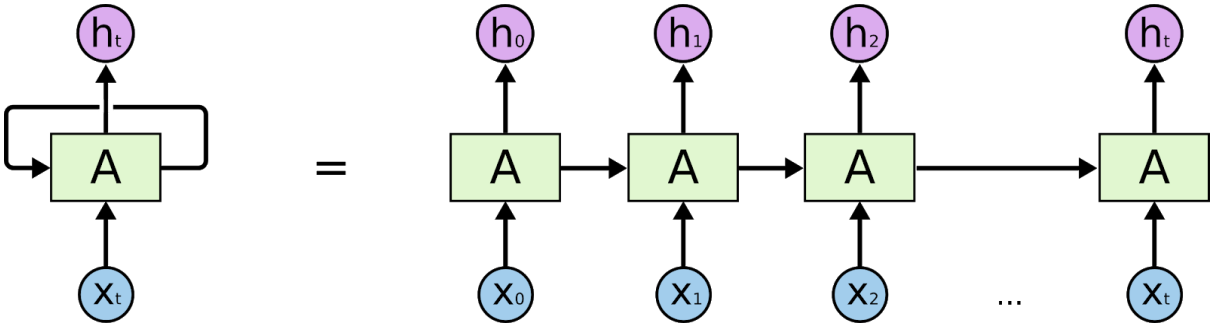
Dominik Kwiatkowski, Jakub Dębski

Opis problemu:

Detekcja anomalii posiada wielorakie zastosowanie w wielu dziedzinach. Służy do identyfikacji nieprawidłowości w sygnałach ciągłych w czasie rzeczywistym. Jednym z przykładów zastosowania jest detekcja próby ataku typu DDOS na serwer, gdy nagle liczba żądań na stronie pikuje. Może być ona także stosowana do wykrywania oszustw giełdowych oraz przy analizie sygnału radiowego w wypadku niwelowania zakłóceń. Głównym problemem detekcji anomalii jest jej zdefiniowanie. Na przykład anomalią może być nagły pik w sygnale, spłaszczenie sygnału, zmiana parametrów sygnału takich jak amplituda czy okres albo nagłe wyrównanie się sygnału. Stąd nie da się jednoznacznie określić czy dane zachowanie jest anomalią, czy też nie jest. W naszym projekcie posłużyliśmy się założeniem, że anomalia jest wtedy, gdy sieć LSTM myli się ponadprzeciętnie, czyli błąd predykcji jest większy od jej średniego błędu.

LSTM:

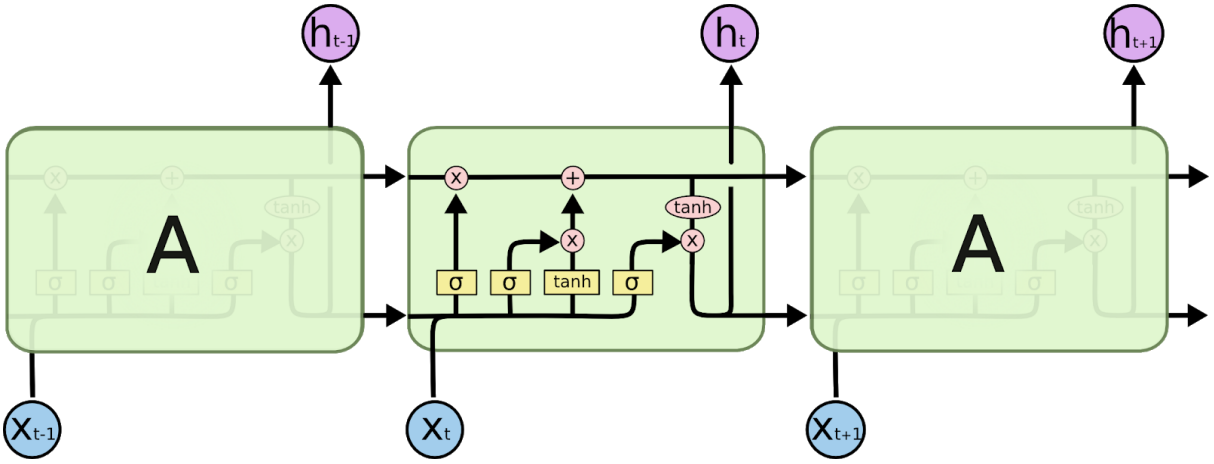
Sieć LSTM wywodzi się z sieci rekurencyjnej, która miała służyć do analizy sygnałów.
Na początku wprowadzimy pewne podstawowe pojęcia. x jest to dana wejściowa. h jest to dana wyjściowa, czyli w naszym przypadku następna wartość sygnału. W przypadku predykcji na podstawie fragmentu czasu sieć po kolei przewiduje wartości.



Grafika 1.1

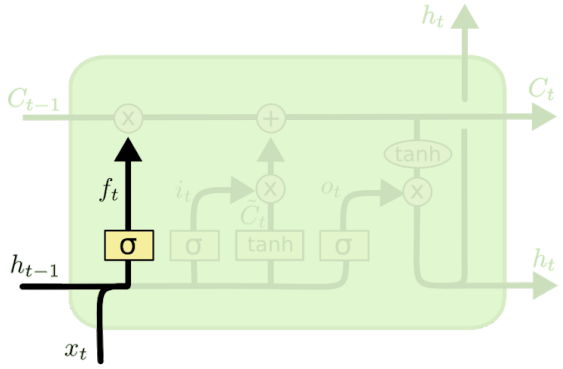
Najprościej to wytłumaczyć na przykładzie predykcji kolejnych słów w zdaniu. W tym przykładzie x to będzie rzeczywiste słowo, a h to będzie słowo przewidywane na podstawie poprzednich zdań. Stąd np mamy zdanie Ala ma i oczekujemy że sieć przewidzi Ala ma kota. Wówczas kolejne inputy to będą kolejne słowa, natomiast kolejne outputy to będa przewidywania następnego słowa na podstawie obecnych. Stąd tak naprawde interesuje nas jedynie ht. Problem jaki niesie klasyczna sieć rekurencyjna to brak wpływu danych odległych. Sieć “pamięta” jedyne kilka ostatnich danych wejściowych co jest

niedopuszczalne dla dłuższych sygnałów. Stąd w roku 1997 zaproponowano lepszy model do predykcji sygnałów.



Grafika 1.2

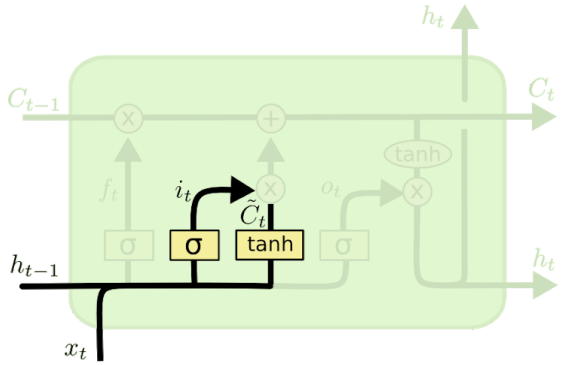
Sieć LSTM jest o wiele bardziej skomplikowana w porównaniu do tradycyjnej sieci neuronowej. Posiada dwie linie przekazywania informacji. Górna to tak zwana cell state. Jest to linia, w której zmiany mogą dokonywać tylko dwie operacje, a ich zakres jest ściśle regulowany przez funkcje. Zawiera ona informacje na temat poprzednich komórek, która po uprzednich modyfikacjach znacząco wpływa na output danej komórki. Kolejne fazy LSTM będę omawiał na przykładzie funkcji sinusoidy o zmiennych właściwościach takich jak okres i amplituda w czasie.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

Grafika 1.3

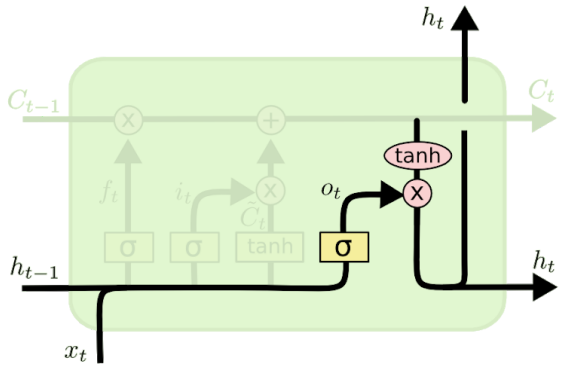
Wskazana bramka to tzw. forgot layer. Jest to pierwsza warstwa neuronowa, w której na koniec uzyskujemy wartości z przedziału 0-1. Mówią one dosłownie, które informacje mamy “zapomnieć”, a które “pamiętać. Dzięki temu funkcja modyfikuje parametry z poprzednich celi w zależności od obecnych danych. Np przy zmianie amplitudy, funkcja powinna zapomnieć wartości z nią powiązane, ale zapamiętać te związane z okresem.



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \; + \; b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \; + \; b_C)$$

Grafika 1.4

Kolejną warstwą jest warstwa dodana nowej wartości. Mamy tutaj 2 warstwy neuronowe. Pierwsza opisana za pomocą funkcji sigmoid nazywana jest input gate layer. Podobnie jak poprzednio służy ona do określenia jakie informacje chcemy dodać.
Wykorzystując poprzedni przykład, chcemy dodać informacje o nowej amplitudzie, ale nie ma potrzeby dodawania informacji o okresie funkcji, który się nie zmienił. Druga warstwa neuronowa wylicza wartości kandydujące, które chcemy wstawić do naszego cell stata. Następnie modyfikujemy wartość z poprzedniego cell stata, mnożąc go przez wektor zapomnienia i dodając nowe wartości kandydujące, uprzednio przemnożone przez input gate.

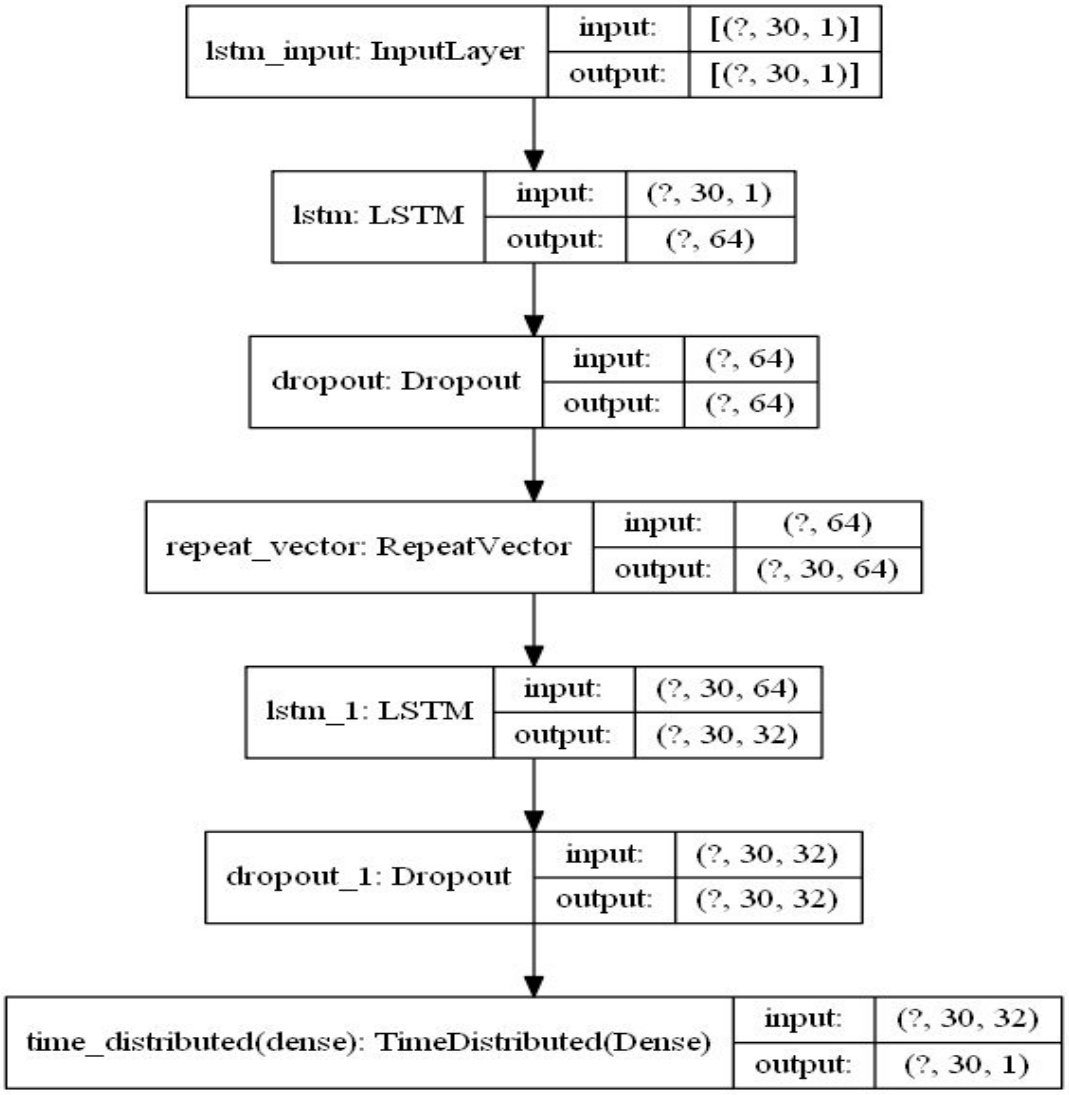


$$o_t = \sigma\left(W_o \; [\, h_{t-1}, x_t] \; + \; b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Grafika 1.5

Zostało nam jeszcze wyliczenie wartości ht. Ponownie za pomocą funkcji sigmoid wyliczamy odpowiedni wektor i przemnażamy go przez stan komórki. Wówczas mamy wartość ht dla danej komórki, którą przekazujemy do następnej komórki oraz dodatkowo przekazujemy stan naszej komórki do następnej. Widzimy, że dolną linią przekazujemy tzw. short memory, czyli wynik z poprzedniej operacji, a górną- stan naszej komórki, który pełni rolę pamięci długodystansowej. Dzięki temu sieć LSTM jest wręcz idealna do uczenia się przebiegu sygnału/funkcji, co mam nadzieję pokażą nasze wyniki.

Model:



Grafika 1.6

Na wejściu naszego modelu dajemy 30 kolejnych wartości poprzedzających wartość, którą chcemy przewidzieć. Następnie przepuszczamy to przez warstwę LSTM, która zwraca jedynie wartości w ostatnim przejściu rekurencyjnym. Kolejny jest Dropout który ma na celu zapobiegać overfittingowi sieci. Robi to poprzez ustawienia losowego elementu inputu na 0 a następnie rozdysponowaniu owej liczby na pozostałe pole inputu tak aby suma całości się nie zmieniła. RepeatVector powoduje powtórzenie naszej operacji. Następnie ponownie puszczamy to przez LSTM, który tym razem zwraca wartości dla każdej sekwencji oraz za pomocą time distributed uzyskujemy wartość wyjściową, czyli nasze ht 30 razy, bo tyle razy podaliśmy input. Podsumowując, na początku uzyskujemy cechy naszego sygnału, potem dodając input i znając jego cechy chcemy na ich podstawie przewidzieć następny dzień. Model wydaje się dość dziwny a jego zastosowanie niekonwencjonalne, bowiem jest on wyszkolony by zwrócić 30 razy tą samą oczekiwaną wartość, tak by ich średnia była jak najbliżej wartości prawdziwej. Model ten powstał w procesie ewaluacji kilku modeli. Każdy z nich zawierał od jednej do kilku warstw LSTM. Niestety, Modele te nie potrafiły dać nam

oczekiwanie małego lossu, byśmy mogli je zastosować. Dopiero po dodaniu Repeat Vector do jednego z nich oraz jeszcze jednej warstwy LSTM przez którą przepuszczamy 30 wartości na podstawie których warstwa time distributed oblicza wartości końcowe dla każdego z wymiarów, udało nam się uzyskać zadowalający nas wynik. Ponadto loss spadł blisko czterokrotnie.

Ewaluacja:

Po znalezieniu odpowiedniego modelu zostały przez nas poddane ewaluacji przede wszystkim 2 czynniki, funkcja losu sieci neuronowej oraz sposób wyliczania granicy błędu, po której uznaje dany przypadek za anomalie. Za pomocą metody prób i błędów oraz obserwowaniu efektu za najlepszą funkcje losu, uznaliśmy mean absolute error. Nasz model przy tej funkcji mylił się znacznie mniej, a błąd dla czystej sinusoidy wynosił około procent, podczas gdy mean percent error miał wynik blisko 10. Również dużo gorzej radziła sobie funkcja logarytmiczna, ze względu na niedoszacowanie wartości. Prawdopodobnie istnieje lepsza funkcja, dokładniej dobrana do konkretnych zagadnień, natomiast wśród naszych danych obrana przez nas funkcja spisywała się zdecydowanie najlepiej. Odmiennie miała się droga ewaluacji threshold. Początkowo chcieliśmy założyć stały procent odchylenia, dla którego stwierdzamy anomalie. Niestety, funkcja różnie sobie radzi dla różnych funkcji i czasem jej zdaniem anomalie nie występowały, a czasem cały wykres był anomalią. Stąd zaszła potrzeba uzależnienia anomalii od możliwości przewidywania. Spróbowaliśmy zatem wziąć średnią z wykresu i tu pojawił się następny problem. W momencie, w którym występowała duża anomalia, wszystkie pozostałe mniejsze piki były nieoznaczalne ze względu na mocne zawyżenie średniej. Stąd postanowiliśmy odciąć skrajne 5% z każdej strony rozkładu Gaussa, w celu wyeliminowania z liczenia średniej skrajnie złych wyników. Wówczas poprawnie wyznaczało nam anomalie na wysokich wartościach, lecz błędnie dla niskich.
Stąd ostateczny wzór na Threshold wyniósł (expected-predicted/expected), czyli wzór na błąd względny. Dodatkowo mnożymy otrzymaną wartość przez 3.5, co zapewnia nam dobre pokrycie anomalii na wykresie.

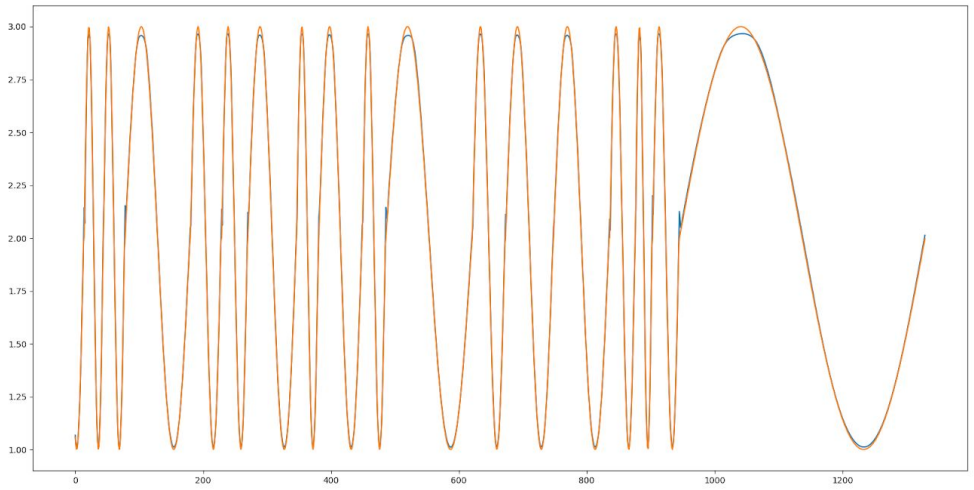
Wykrywanie anomalii:

Finalnie nasz wzór na błąd to abs(avg(model.predict)-expected)/expected. Innymi słowy bierzemy średnią wartość z 30 wartości zwróconych przez sieć. Następnym krokiem jest odjęcie wartości prawdziwej i wzięcie wartości bezwzględnej z owej różnicy. Następnie dzielimy otrzymany wynik przez wartość prawdziwą aby otrzymać procentowy błąd względem wartości prawdziwej. Nasz wzór na Threshold to średnia wartość błędu * stała, przy czym od średniej usunęliśmy 5% skrajnie dobrych i skrajnie złych wartości. Natomiast dany punkt jest uznawany przez nas za anomalię wtedy gdy błąd procentowy wyliczony z powyższego wzoru jest większy od ustalonego Thresholdu.Ważną zależnością jest to, że zakładamy że nasz model predykcja wartości bliskie idealnym. Im większy jest błąd sieci, tym gorzej ona się sprawuje. Szczególnie widoczne jest to przy giełdzie, kiedy praktycznie niemożliwe jest stwierdzenie anomalii, bowiem istnieją zakresy, na których sieć sobie

kompletnie nie radzi. Stąd przed użyciem LSTM do wykrycia anomalii należy być pewnym, że sieć potrafi przewidywać sygnał w sposób bardzo dobry, a jeśli nie potrafi, to wykrywanie anomalii się nie uda.

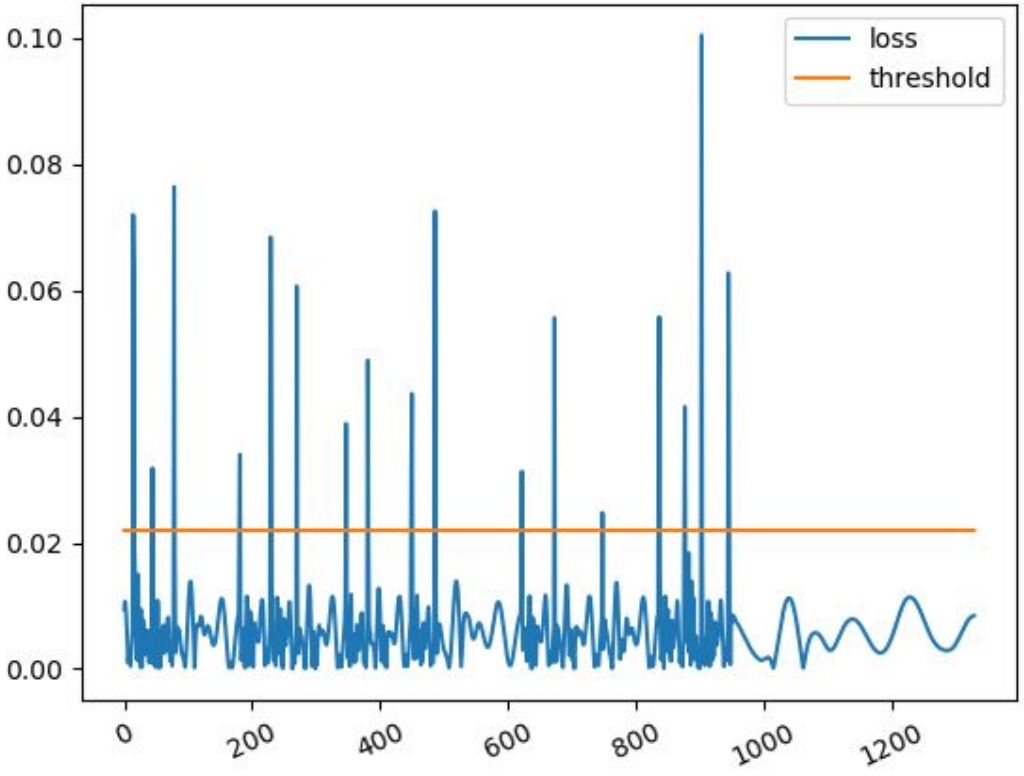
Sinusoida:

Naszym zadaniem było sprawdzenie jak LSTM poradzi sobie z różnymi sygnałami oraz anomaliami w nich. Na początek postanowiliśmy ręcznie wygenerować sygnał w postaci sinusoidy o zmiennych losowych okresach. Najpierw warto zobaczyć, jak dobrze LSTM jest w stanie nauczyc sie dość prostego sygnału.



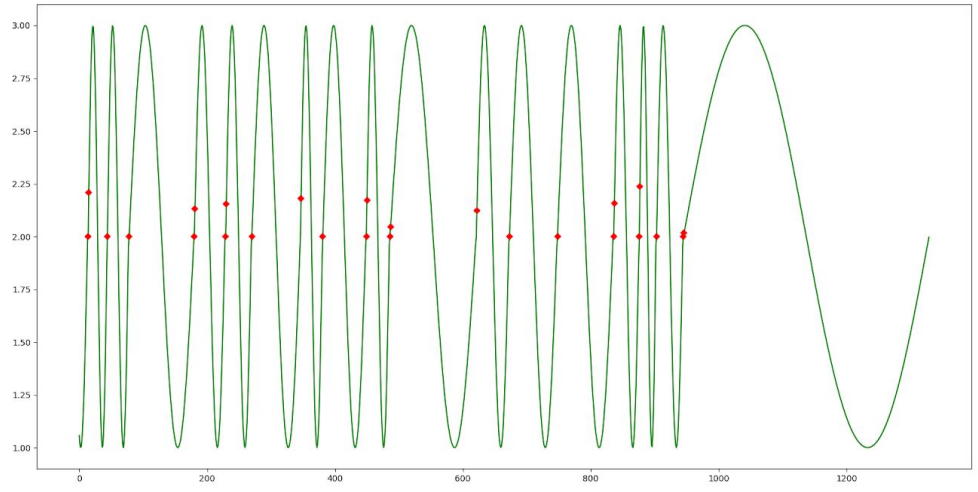
Grafika 2.1

Przedstawiony tutaj wykres różnić(niebieski przewidywana, pomarańczowy spodziewane) są dosyć zbliżone. Jedyne zauważalne różnice występują w momencie zmiany okresu, jednak funkcja bardzo szybko adaptuje się do zmiany sygnału oraz na brzegach sinusoidy ze względu na to, że funkcja próbuje przewidzieć ewentualną zmianę okresu. Algorytm szukania anomalii wyglądał następująco. Dla każdej próbki obliczaliśmy różnice pomiędzy sygnałem wygenerowanym przez funkcje a wartością przewidzianą przez LSTM, następnie obliczyliśmy średnią różnice po odrzucaniu skrajnych 5% rozkładu Gaussa. Uzyskany wynik został przemnożony przez pewną stałą, dobieraną w zależności od tego jak duże rozbieżności chcemy zaliczać jako anomalie, a następnie zaznaczyliśmy na wykresie punkty, dla których błąd był większy od wyliczonej wartości odcięcia.



Grafika 2.2

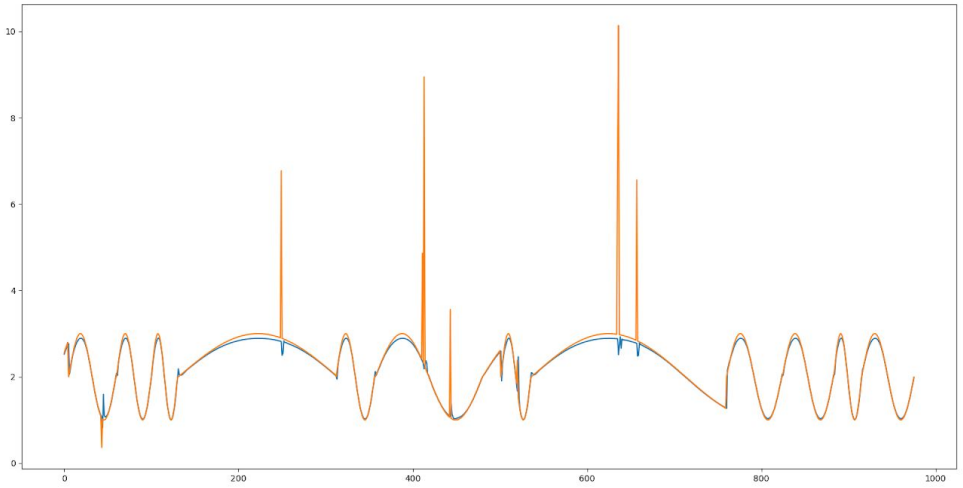
Na tym wykresie widzimy linie progu oraz wykres błędu w całej dziedzinie. Warto zauważyć, że dla większości przedziału błąd jest poniżej 0.01, co stanowi świetny rezultat.



Grafika 2.3

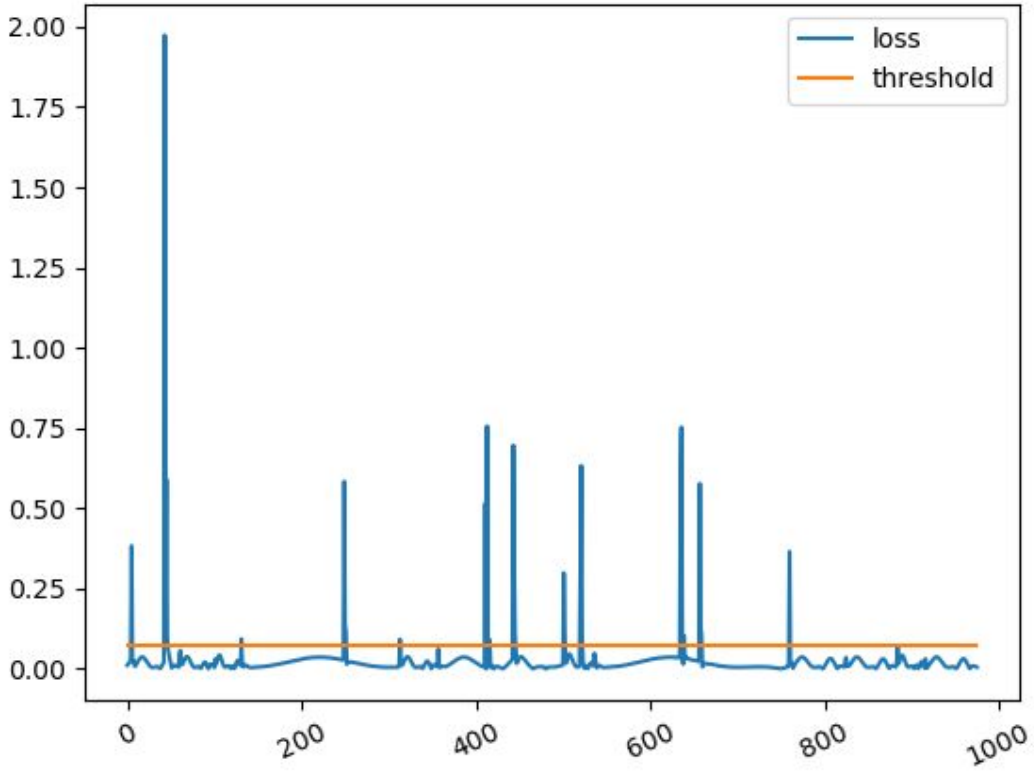
Tutaj mamy wykres z anomaliami. Widzimy, że nasz algorytm wyznaczył jako punkty anomalii tylko miejsca zmiany okresu. Jako, że nowy okres jest randomizowany, LSTM musi

się pomylić w tym miejscu, bowiem wynik jest nie do przewidzenia. Ze względu na obraną metodę obliczania progu wyznaczania anomalii, nie jest możliwe wyeliminowanie tak dużego względem normy błędu. Natomiast moim zdaniem zmiana okresu dla sinusoidy spełnia definicję anomalii, stąd można uznać, że nasz algorytm działa poprawnie. Zmodyfikujemy teraz naszą funkcje generująca sinusa o wstrzykiwanie anomalii. Szansa na anomalie wynosi 1%, a anomalia to zrandomizowane pomnożenie wartości przez odpowiednio duży współczynnik.



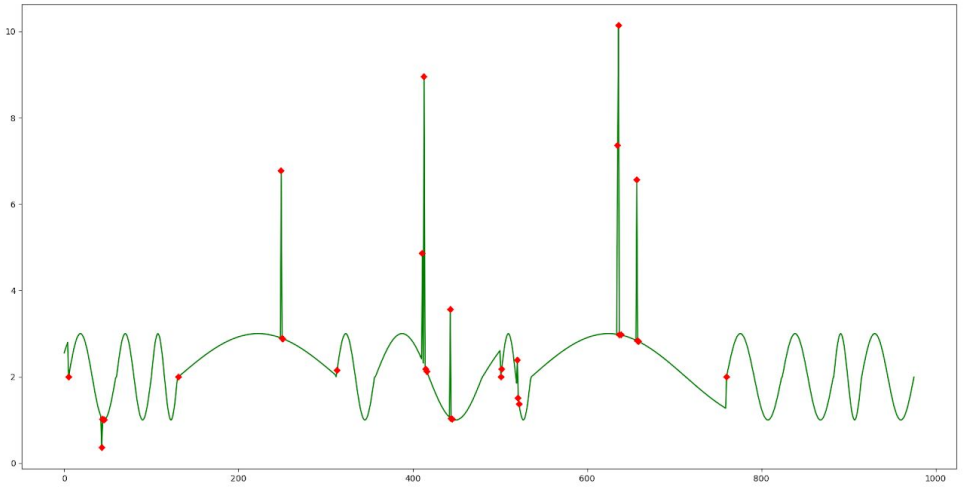
Grafika 2.4

Jak widać anomalia jest niemożliwa do spredykowania, natomiast sieć neuronowa myli się przez kilka punktów po jej wystąpieniu. Wynika to z faktu nietypowego zachowania nierozpoznanego przez model. Natomiast ogólne wyniki są zadowalające gdyż po wystąpieniu anomalii, błąd stosunkowo szybko maleje do wartości poniżej thresholdu.



Grafika 2.5

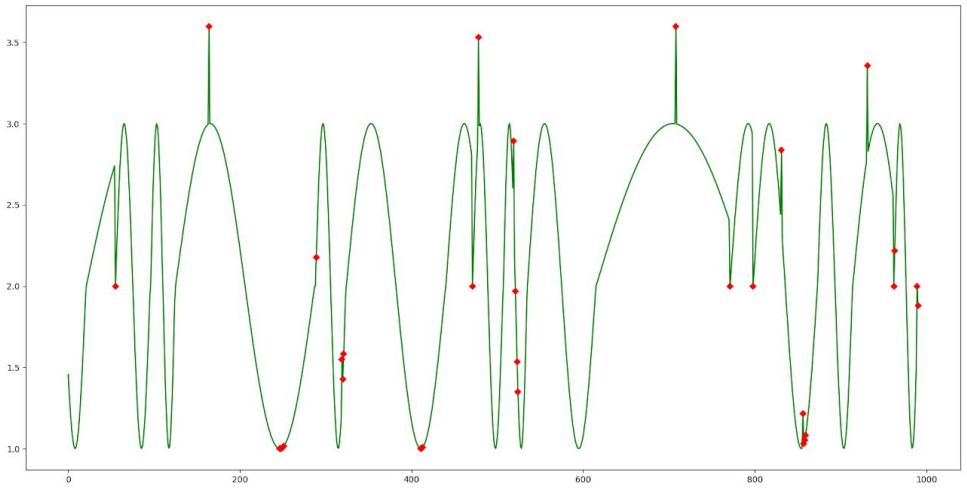
Na wykresie odcięcia widać wyraźnie momenty wystąpienia naszych anomalii,



Grafika 2.6

Nasz algorytm poprawnie oznaczył wszystkie anomalie. Dodatkowo oznaczył 1-2 następne punkty, niemniej wskazanie anomalii działa poprawnie. Warto zauważyć, że tym razem oznaczył tylko największe załamania wykresu, co wiąże się ze wzrostem progu oznaczenia

anomalii. Na koniec postanowiłem utrudnić drogę algorytmowi i zmniejszyć wielkość anomalii.

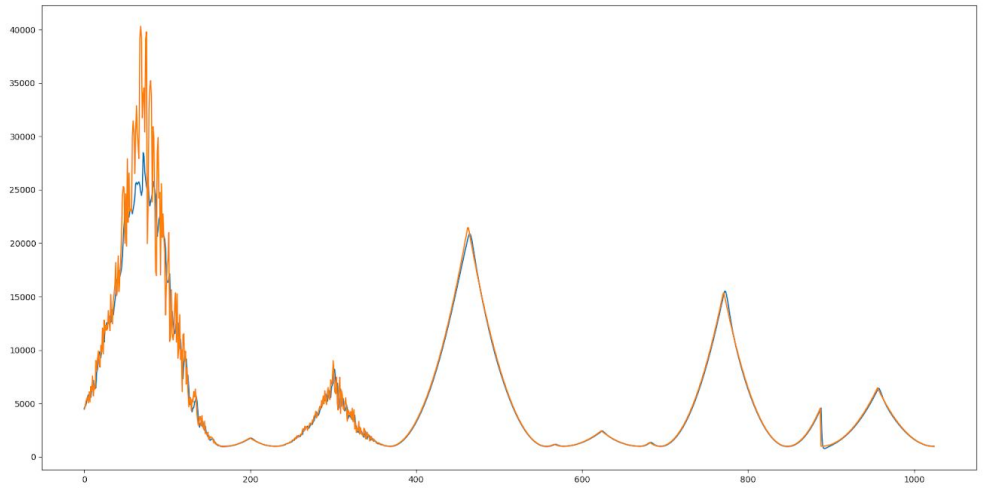


Grafika 2.7

Ponownie algorytm znalazł wszystkie anomalie i oznaczył kilka punktów załamań sinusiody. Wnioskuje z otrzymanych rezultatów, że wykrywanie anomalii dla regularnych sygnałów za pomocą metody LSTM jest efektywne.

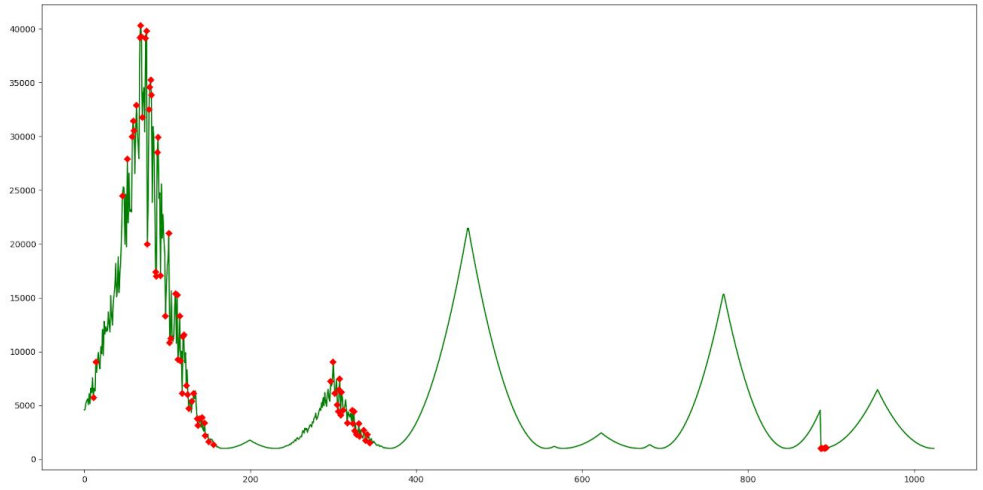
Sinusoida z wielomianem:

Po temacie funkcji sinusa ze zmiennym okresem podjeliśmy badanie nad funkcją nieco mniej przewidywalną. Utworzyliśmy funkcję losującą wielomian określonego stopnia, w tym przypadku 4, oraz jego współczynniki, również z określonego przez nas zakresu. Kolejnym etapem randomizacji funkcji było przydzielenie jej określonego okresu z pewnego przedziału. Po określeniu podanej funkcji, utworzyliśmy jeszcze dwa dodatkowe losowo występujące zdarzenia. Pierwsze z nich polegało na przerwaniu okresu funkcji w połowie, drugie natomiast na symulacji szumu radiowego przez przemnożeniu wartości przez losową wartość z przedziału.



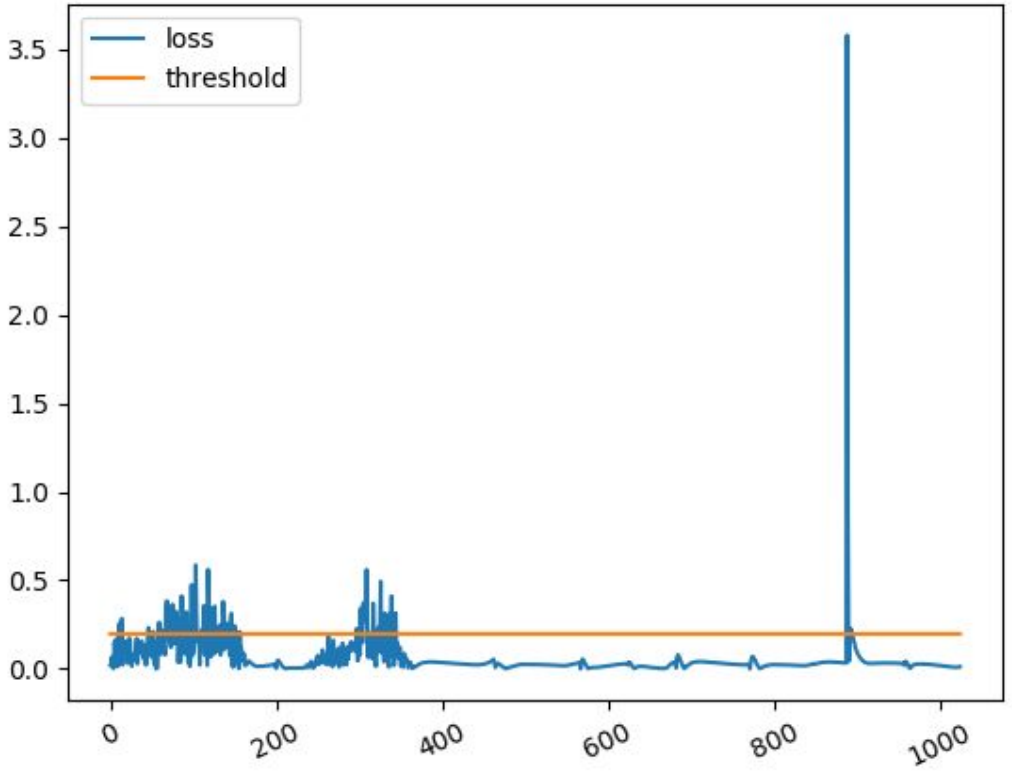
Grafika 3.1

Jak widać na powyższym wykresie, wartości predykowane pokrywają się z wartościami faktycznymi funkcji. Jedynymi widocznymi wyjątkami są naniesione anomalie w postaci szumu radiowego i zakończeniu okresu funkcji w połowie okresu (w okolicy 900 punktu).



Grafika 3.2

Sieć bardzo dobrze radzi sobie ze zmiennymi okresami oraz współczynnikami. Przy mniejszych losowościach szumu radiowego sieć potrafi trafnie przewidzieć wartości. Największy procentowo błąd dzieje się w momencie odcięcia funkcji w połowie okresu. Niemniej sieć bardzo szybko zaadaptowała się do zmienionej sytuacji i poprawnie przewidziała ostatni okres.

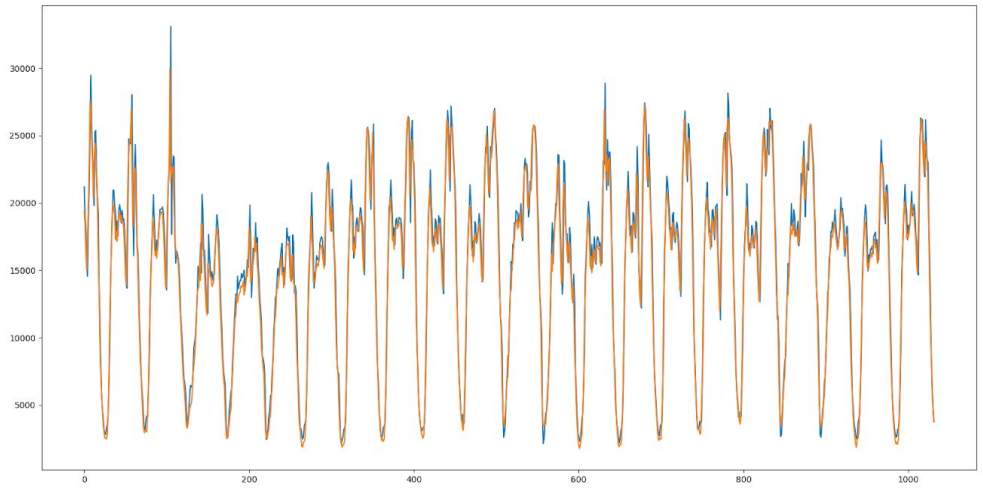


Grafika 3.3

Procentowe wyniki różnicy są więcej niż zadowalające. W przypadku wystąpienia zakłóceń sieć popełniała błąd analogiczny do wprowadzonego współczynnika, przez który losowo mnożyliśmy. Największy pik, jak wcześniej powiedziano, został spowodowany nagłym przerwaniem okresu funkcji. Nie jest on jednak brany do obliczania progu odcięcia z powodu zaimplementowanej funkcji, która odcina skrajne wartości z obu stron krzywej Gaussa.

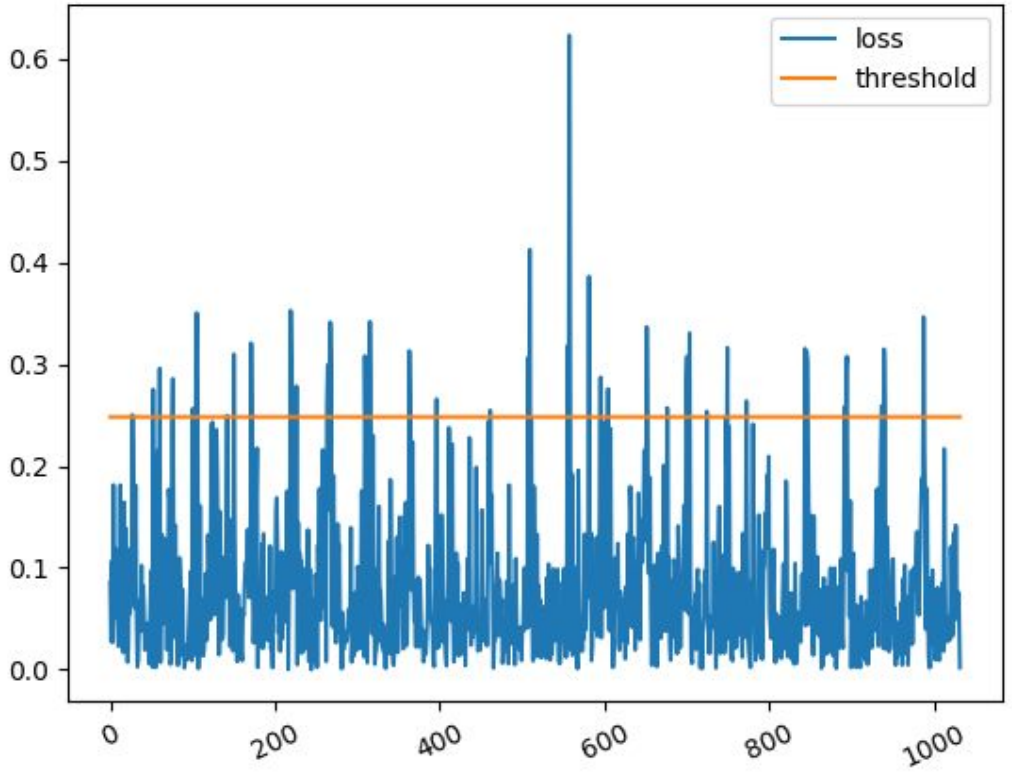
Taksówki, Nowy Jork:

Następnym zbiorem danych wykorzystywanym przez nas do uczenia sieci LSTM były dane archiwalne z zapotrzebowania na taxówki w obrębie miasta Nowy Jork w zależności od godziny. Problemy które napotkaliśmy w trakcie realizacji tego podpunktu polegały na nieregularności datasetu, głównie ze względu na to że ww. zapotrzebowanie zależy również od innych składowych, których nie wykorzystaliśmy (np. pogody). Mając 10000 punktów wykorzystaliśmy 9000 do utworzenia modelu, a następnie predykowaliśmy pozostały 1000, sprawdzając przy tym o ile wartość przewidziana myliła się od stanu faktycznego.



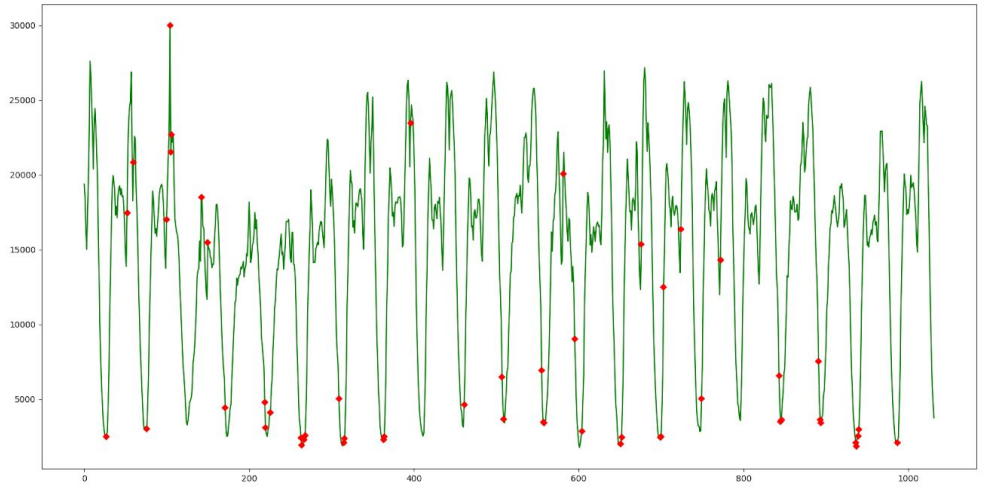
Grafika 4.1

Gołym okiem widać że największe rozbieżności między danymi uzyskanymi przez sieć a prawdziwymi były w momentach, w których wartość drastycznie się zmieniała. Jest to uwarunkowane faktem, iż sieć nie radzi sobie najlepiej w przypadku zmiany drugiej pochodnej. Niemniej jednak, sieć zauważyła “pattern” zapotrzebowania od czasu.



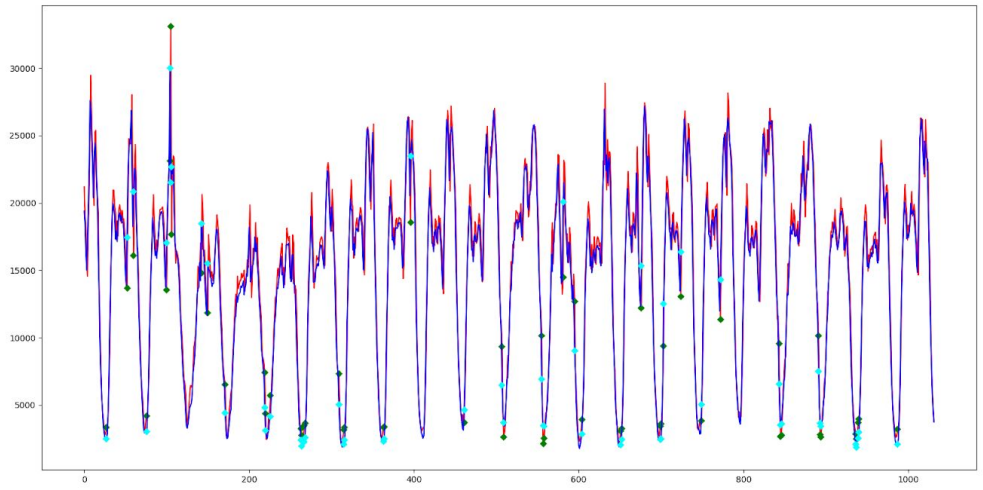
Grafika 4.2

Wykres stosunku różnicy wartości przewidzianych i wartości prawdziwych jest znacznie bardziej pofalowany od wykresu sinusa w którym dużo lepiej było widać wszelkie nieprawidłowości. Podobnie jak w przypadku sinusa, wykorzystaliśmy tutaj wyliczanie progu na podstawie odcięcia 5% skrajnych w krzywej Gaussa, a następnie wyliczenie z tego błędu średniego. Kolejno przemnożyliśmy to przez stałą, w tym przypadku 3, aby podnieść wartość progu, dzięki czemu po górnej stronie thresholdu zostały już tylko piki, które moglibyśmy traktować jako anomalie.



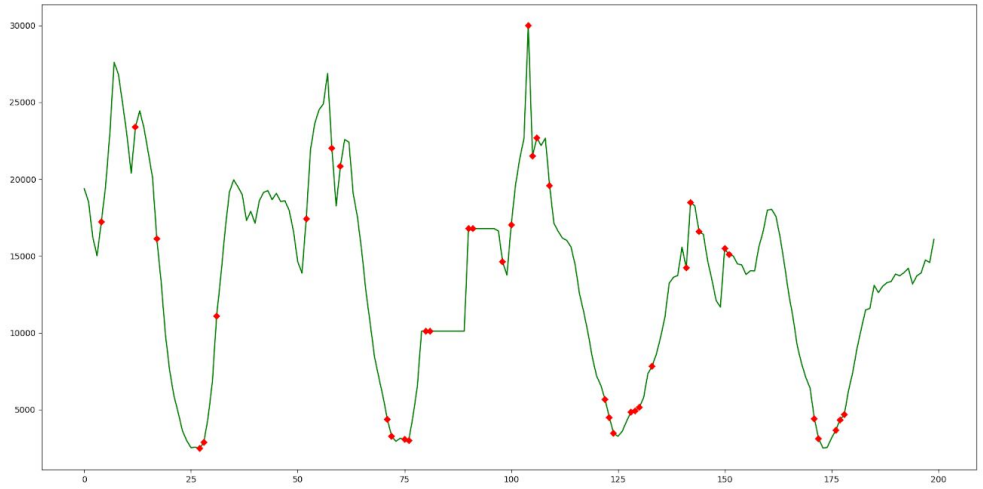
Grafika 4.3

Ze względu na fakt, iż wartość błędu była liczona na podstawie jego wartości liczbowej, która została podzielona przez wartość faktyczną błędu, Anomalie w dolnej części wykresu są znacznie regularniejsze, ponieważ w przypadku dzielenia przez wartości z dolnego piku ten sam błąd numeryczny stanowi większą procentową część wartości. Tak jak w przypadku sinusoidy, anomalie często występują “w grupach” po 2-4 punkty. Niemniej widać, że w przypadku nieoczekiwanych załamań algorytm poprawnie znajduje nieregularności. Ponadto, warto usprawiedliwić dokładność algorytmu małą ilością danych.



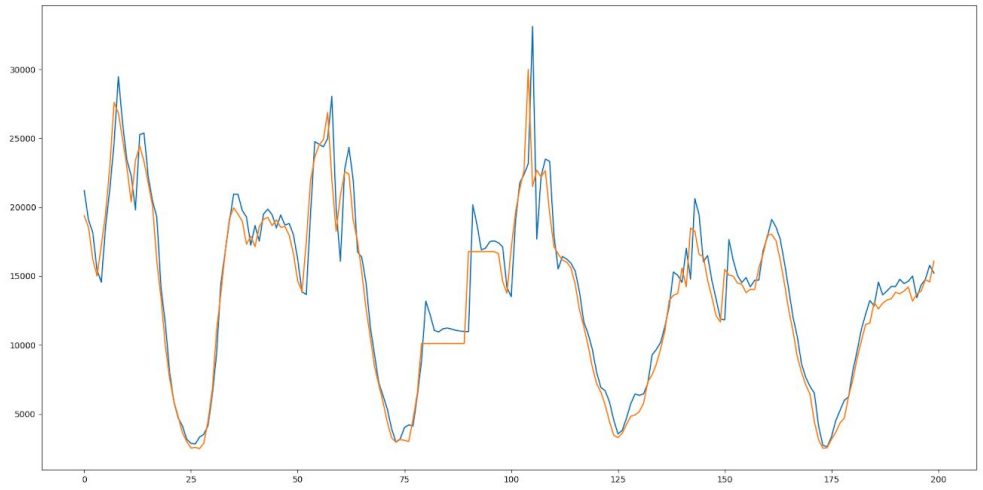
Grafika 4.4

Kolejnym eksperymentem, który przeprowadziliśmy była punktowa manualna modyfikacja sygnału, polegająca na wypłaszczeniu krzywej.



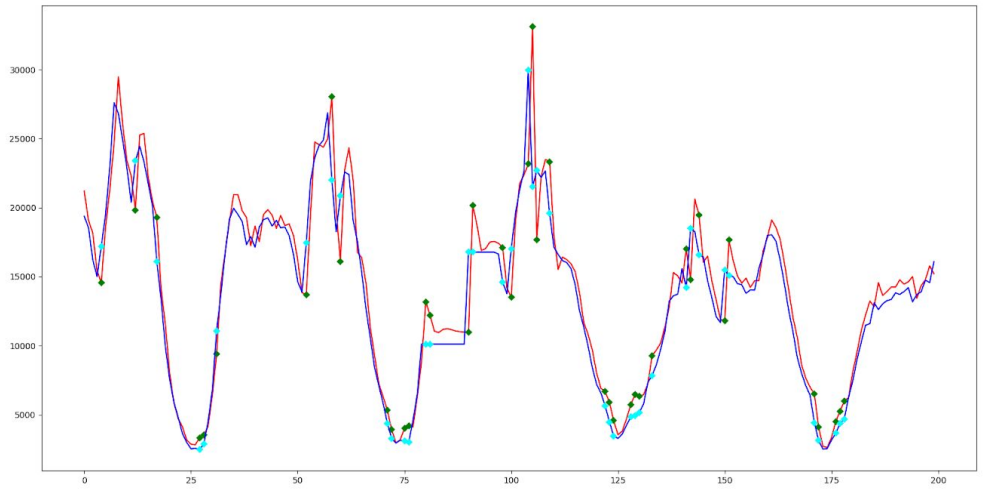
Grafika 4.5

W okolicach 80 node zostały wprowadzone manualnie dwa wypłaszczenia na poziomie logicznym dla sieci. Sieć poprawnie zlokalizowała sam początek anomalii, po czym starała się dostosować swoje wyniki do zmieniającej się sytuacji.



Grafika 4.6

W pierwszym punkcie manualnej zmiany, sieć zauważyła anomalie, a następnie szybko wypłaszczyła swoje estymaty. Efektem owego spostrzeżenia sieci był błąd, który zmniejszył się do niecałych 10% wartości, co jest wartością poniżej thresholdu.



Grafika 4.7

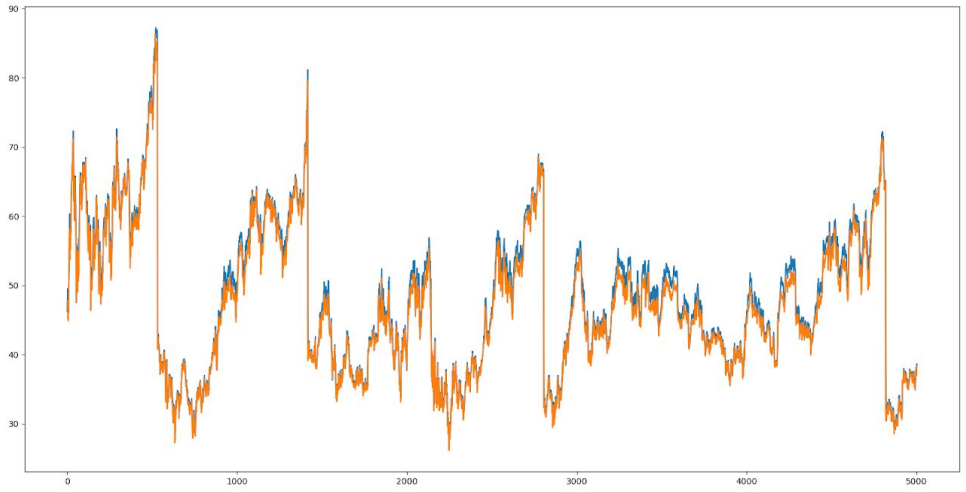
Wartym zauważenia jest również fakt, iż przy wypłaszczeniu na poziomie 16000 w okolicach 90 punktu sieć bardzo szybko zareagowała kontrując wartość do odpowiedniej. Może to wynikać z nieregularności danych, na których była uczona i doskonałym przystosowaniu sieci LSTM do powyższego zadania. Dużą rolę odgrywa dobre wyuczenie się tej anomalii przez forgot layer. Prawdopodobnie jest w stanie “zapamiętać” moment wypłaszczenia i szybko dostosować się do niego. Również widać, że funkcja bardzo szybko wraca do normy

po ustaniu wypłaszczenia, co również świadczy o dobrej znajomości przez sieć tego momentu.

Podsumowując, sieć dobrze radzi sobie nie tylko w laboratoryjnie wygenerowanych przypadkach takich jak modyfikowany sinus, lecz również w przypadku danych ze świata realnego. Sieć jest w stanie przewidzieć z pewną trafnością sygnały i na ich podstawie jesteśmy w stanie wykryć ewentualnie nieprawidłowości w działaniu chociażby ruchu lądowego.

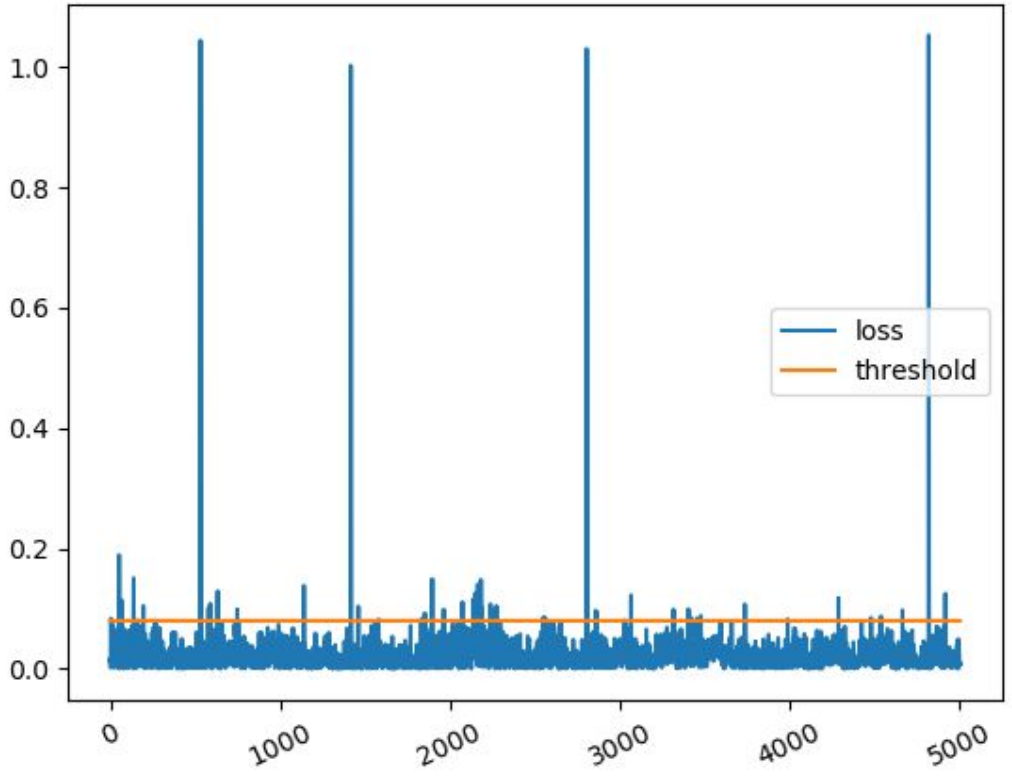
Giełda:

Ostatnią i najtrudniejszą dla sieci próbą, jest próba zrozumienia giełdy. Giełda jest bardzo złożona i jej cena jest dla człowieka nieprzewidywalna do tego stopnia, że w 40 letnim eksperymencie małpa zarobiła na giełdzie więcej niż najlepsze fundusze indeksowe.



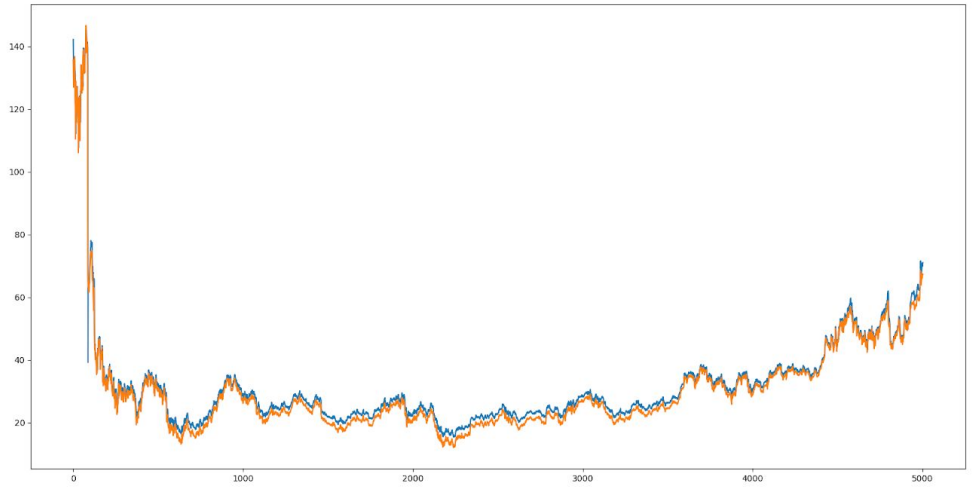
Grafika 5.1

Warto dodać, że sieć była uczona na 1000 spółek giełdowych. Jak widać, mniej więcej połowa okresu jest dobra, a druga połowa zdecydowanie gorsza. Lepiej powinien to zobrazować threshold.

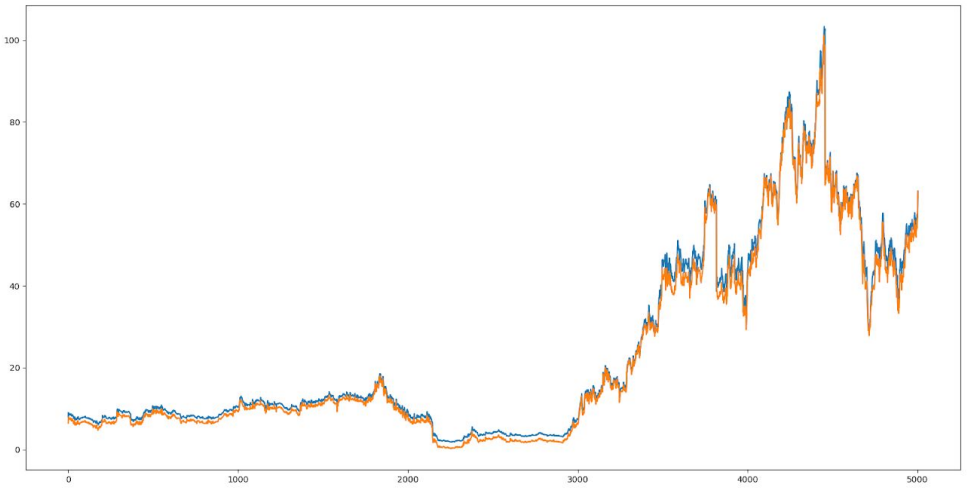


Grafika 5.2

Z powyższego wykresu widać wyraźnie grupy, w których sieć działa lepiej oraz grupy w których działa gorzej. Jest to szczególnie widocznie za nodem 2000. Podobny błąd można zauważyć na wykresach innych firm.

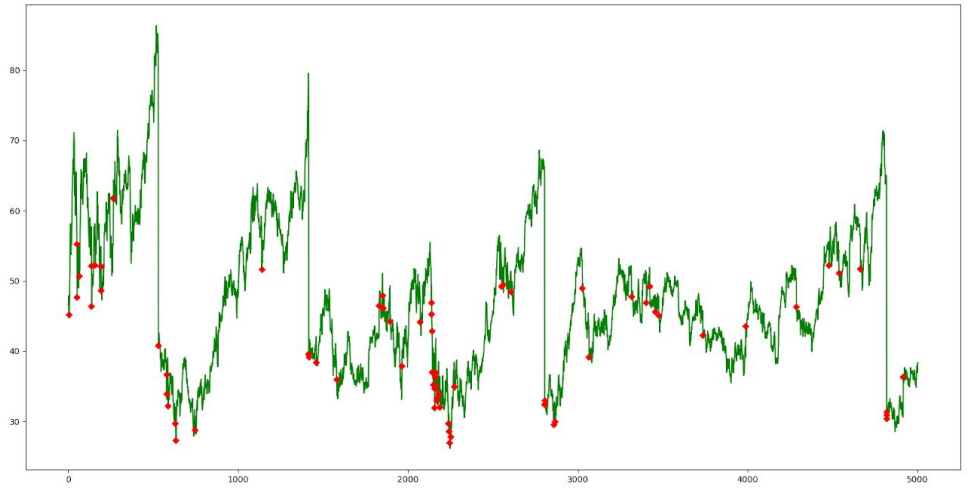


Grafika 5.3



Grafika 5.4

Przez to, znalezienie anomalii może być błędne.



Grafika 5.5

Jak widzimy duże piki są oznaczane, sieć kompletnie nie jest wstanie ich przewidzieć oraz dodatkowo, łapie całe grupy błędów, które oznacza jako anomalie. Są więc dwie opcje, albo jest to faktyczna anomalia na wykresie, czyli na przykład oszustwo giełdowe. Natomiast zakładam, że skoro komisja giełdowa nie orzekła takowego, to jest to prawidłowe zachowanie giełdy. Zatem sieć LSTM nie jest w stanie przewidywać giełdy na tyle dokładnie by móc orzekać o anomalii na niej bądź jej braku. Natomiast mając prawdziwy wykres giełdy może wskazać obszary podejrzane, które zachowywały się atypowo, co może być pomocne w poddawaniu selekcji danych do dokładniejszego prześledzenia przez komisje, które zajmują się ochroną giełdy przed oszustwami. Dzięki temu może ułatwić to prace komisji papierów wartościowych i innych organizacji tego typu. Wymagałoby to natomiast

porównania jak sieć działa na akcjach, które faktycznie były użyte do oszustwa, co niestety jest tajne i niemożliwe do uzyskania przez nas.

Podsumowanie:

Model sieci rekurencyjnej LSTM posiada świetne zdolności w zakresie przewidywania danego sygnału, nawet pomimo losowości części parametrów. Dodatkowo sieć szybko kontruje wystąpienie anomalii takich jak piki oraz inne atypowe zachowania funkcji. Znajdowanie anomalii w stałym ciągu jej pomocą może mieć wielorakie zastosowanie biznesowe. Do głowy przychodzą mi pomysły w monitorach medycznych pracy narządów, w których sygnał jest stały, a sieć po wykryciu anomalii mogłaby zawiadomić służby medyczne. Warunkiem dobrego działania LSTM są duże rozmiary danych. Podwojenie danych z naszych obserwacji zmniejsza błąd przewidywania o 30-40%, stąd moim zdaniem sieć jest w stanie nauczyć się dowolnie trudnych sygnałów, pod warunkiem jakiejkolwiek ich zależności. Niestety, giełda jest zbyt skomplikowana i działa na nią zbyt wiele czynników, bowiem użycie kilkudziesięciu milionów timestampów nie dało oczekiwanego rezultatu. Natomiast dla sygnałów o pewnej regularności, LSTM bezproblemowo zaadaptuje się do zmian. Moim zdaniem obszarem, w którym LSTM by się sprawdził jest obrona przed atakami typu DDOS. Sieć bez problemu zaadoptuje wzrost użytkowników, który będzie ciągły w czasie, natomiast oznaczy jako anomalie nagły wzrost zapytań spowodowany atakiem na serwer. Te i inne zastosowania sieci mają duże znaczenie w warstwie biznesowej, stąd uważam ten model za istotny w dziedzinie sztucznej inteligencji. Natomiast wadą sieci LSTM jest duża podatność na nieznane przypadki. Kiedy do predykcji podawaliśmy dane, które zawierały “nowe” anomalie, wówczas sieć dosłownie ogłupiała i zwracała wartości z jeszcze większym odchyleniem od normy niż anomalia. Stąd sieć LSTM musi uczyć się na sygnale, który zawiera anomalie. Dodatkowo te anomalie muszą być rzadkie, by LSTM nie próbował ich przewidzieć, ale umiał zachować się po ich wystąpieniu.

Źródła:

https://colah.github.io/posts/2015-08-Understanding-LSTMs/
https://towardsdatascience.com/anomaly-detection-with-lstm-in-keras-8d8d7e50ab1b
https://en.wikipedia.org/wiki/Long_short-term_memory
https://github.com/numenta/NAB/tree/master/data
https://github.com/cerlymarco/MEDIUM_NoteBook/blob/master/Anomaly_Detection_LSTM/nyc_taxi.csv
https://www.pb.pl/malpy-lepsze-na-gieldzie-niz-fundusz-indeksowy-711896
https://towardsdatascience.com/step-by-step-understanding-lstm-autoencoder-layers-ffab055b6352
https://www.tensorflow.org/api_docs/python/tf
http://www.bioinf.jku.at/publications/older/2604.pdf