タイトル

1. 研究背景

1.1 ディープラーニング（深層学習）

従来型の時系列予測の研究に関しては主に統計分析手法と計量経済学モデルを用いて行っているが、1980年代以降、金融市場の分析に機械学習の手法が徐々に適用されるようになった。しかし、複雑な関数近似をしなければ分類や回帰ができない場合があり、従来型の機械学習手法ではうまくいかないケースも多く、このような問題に対して、ディープラーニング（深層学習）手法を使用するケースが増えている。ディープラーニングの手法を用いることにより、従来に比べて飛躍的に認識精度が向上するケースもあり、現在世の中でディープラーニングはたいへん注目されている。例えば、最近ではレコメンデーションや自動運転の分野など、幅広い分野で利用されている。ディープラーニングの領域の中でベースの手法はニューラルネットワークと呼ばれる機械学習手法である。

1.2 ニューラルネットワーク

ニューラルネットワークの誕生からディープラーニングへの発展を簡単に振り返ると、その起源は1943年にまで遡る。この年に「形式ニューロン」という人間の脳を模した数理モデルが提唱され、1957年にはデータによる学習が可能な「パーセプトロン」というニューラルネットワークが開発される。パーセプトロンは1960年代に一大ブームを引き起こすが、マービン・ミンスキーにより弱点を指摘され、急速に勢いを失い、冬の時代を迎える。ニューラルネットワークは、何度か、このようなブームと冬の時代を繰り返すが、2006年のトロント大学のヒントン教授による技術革新と、同時期に画像認識コンペティションでの圧倒的な認識精度での優勝という出来事があり、3回目のブームを迎えることになる。

ニューラルネットワークは脳の神経回路をモデルした機械学習手法の一つである。生きもの脳はニューロンと呼ばれる細胞が膨大な数集まって構成されている。そしてそれぞれのニューロンが電気信号を受け取ったり、出力したりして脳は活動している。ニューラルネットワークの場合はモデル化されたニューロンが複数の入力を受け取って、それらの値を重み付きで足し合わせて、出力する。一つひとつの人工ニューロンは単純な仕組みだが、それを多数組み合わせる事で複雑な関数近似を行うことができる。

ニューロンを横に並べたり多層化したものがニューラルネットワークになる。ニューラルネットワークの各層には名前がある。入力データが入って来る層のことを入力層、最終的に求めたい値が出てくる層を出力層、それ以外の中間にある層のことを中間層と呼ぶ。

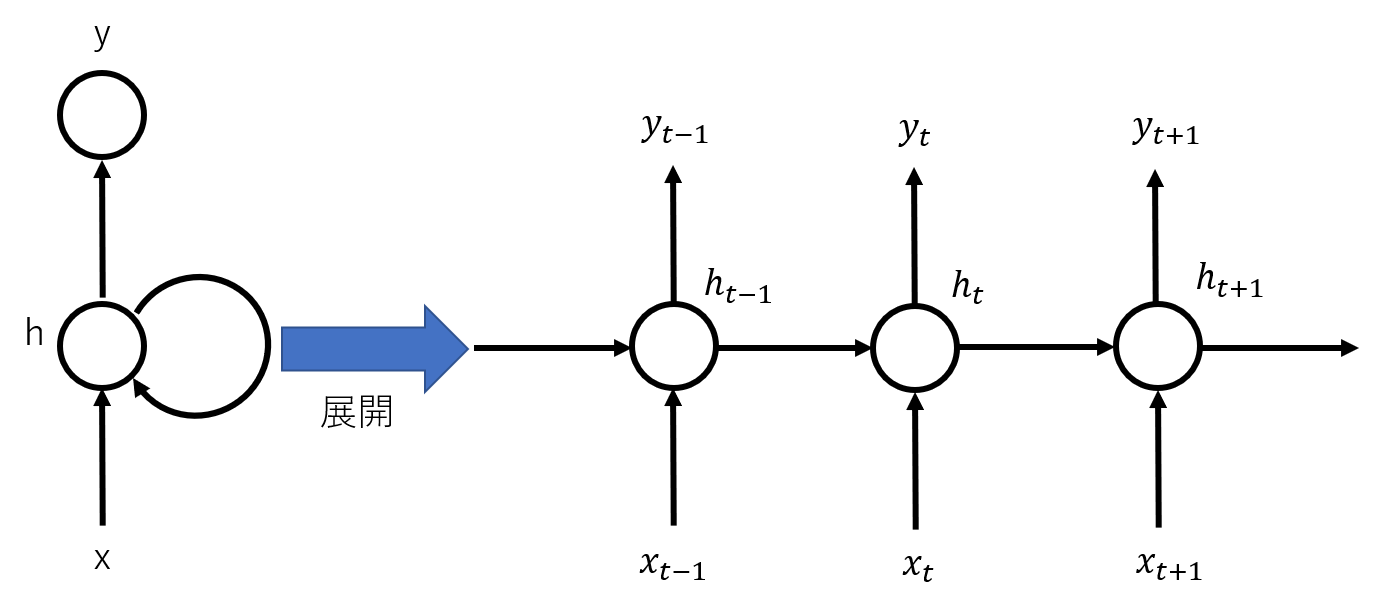
しかし、時系列データの分析に関しては、入力は一つひとつ一連の入力データとして扱われ、それぞれに対する一連の出力データが必要となる。単純なニューラルネットワークでは、出力は出力層からだけ発生するため、一連の入力一つひとつに対して出力を生成することはできない。通常のニューラルネットワークでは、ある層の出力は、次の層の入力に利用されるのみであり、時系列データを自然に扱うことができない。

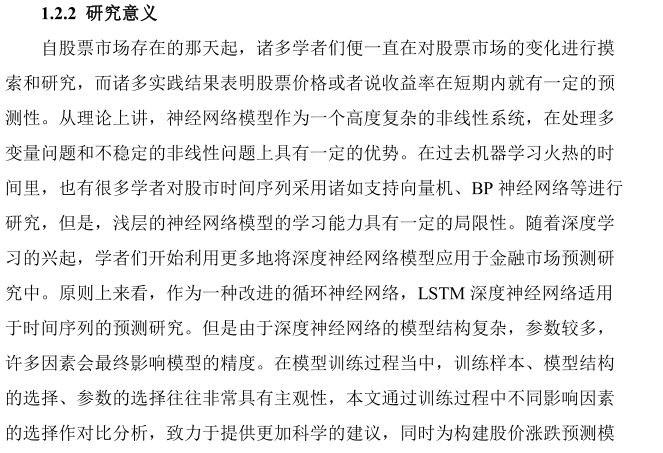
二、研究方法

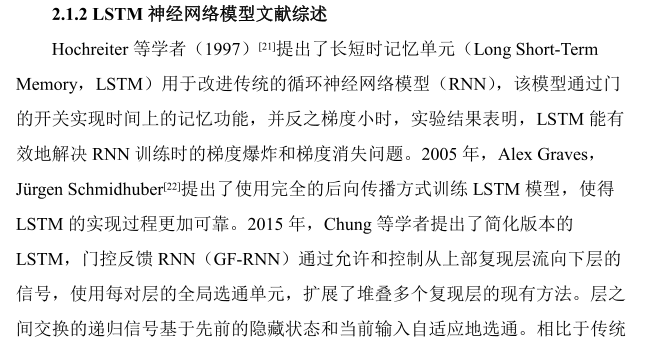
2.1　再帰型ニューラルネットワーク

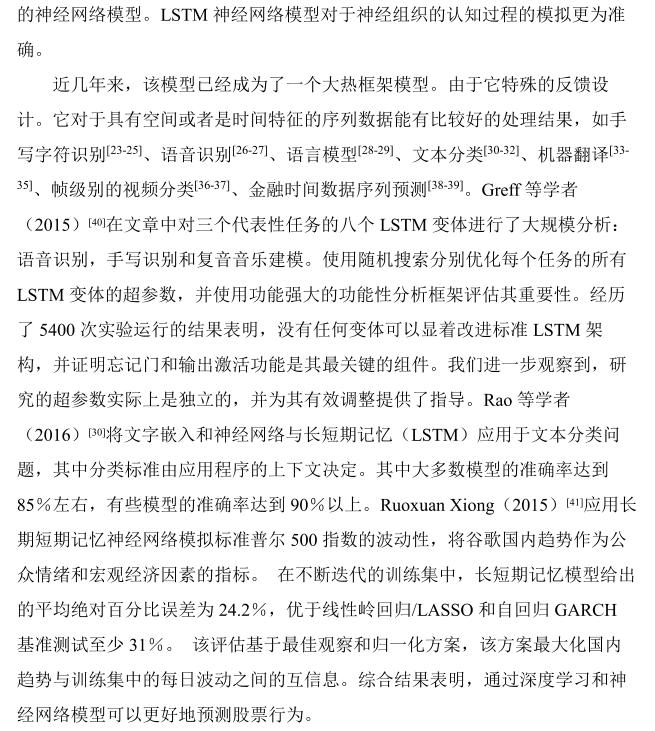
再帰型ニューラルネットワーク（Recurrent Neural Network：以下、RNN）では、層自体は入力層、中間層、出力層と一般的なニューラルネットワークと変わらないが、時刻ｔにおけるｘｔに加え、時刻ｔ－１における中間層の値を保持しておき、それも時刻ｔにおける中間層に伝える点が異なる。時刻ｔの状態をｔ－１の状態として保持し

ある層の出力は、次の層の入力として利用されるだけでなく、一般的なニューラルネットワークの出力層のような出力としても利用される。また、各層の入力として、前の層の入力のみではなく、時系列のデータポイントも入力とする。だから、RNNは時系列データの予測問題を解決することができる。

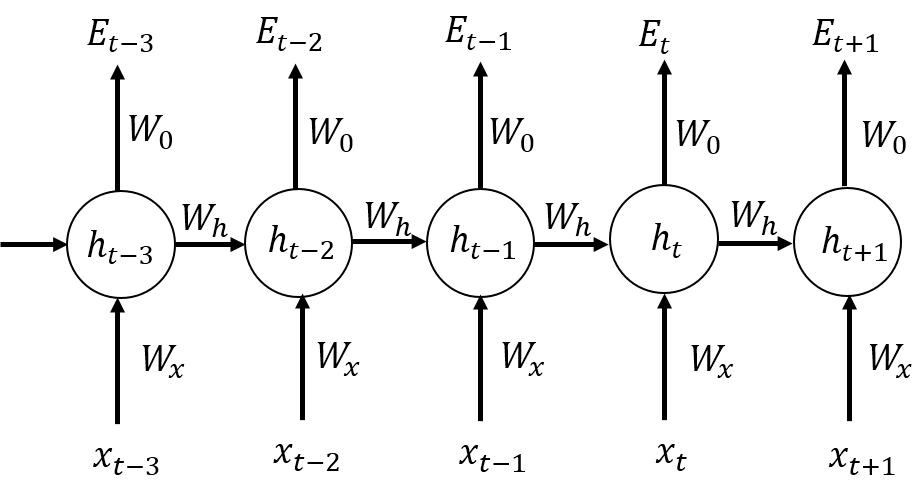








Back Propagation Through Time(BPTT)



RNNの概要は前の図　で示した通り、

はｔ時刻の入力

はｔ時刻の中間層の出力

は誤差

は入力層から中間層の重み

は中間層同士の重み

は中間層から出力層への重み

時刻ｔの中間層の計算をする際に入力層からの入力と前の時刻の中間層の出力に重み付けを行った結果を加算したものを中間層への入力として扱う。この際にポイントが2つある。

一つ目は、一般的なRNNでは時刻のｔ＝0のときには時刻t-1の中間層が存在しないため、過去の時間からの入力はすべての値が0のものを時刻t-1の中間層からの入力として設定する。

2つ目は、入力層から中間層への重み、中間層から中間層への重み、中間層からその上の層への重みは別の時間軸関でそれぞれ同じものを用いる。すなわち、一つのニューラルネットワークを構築して、その中間層の計算にとある特定の重みを用いて時間展開をしていくということになる。その際の活性化関数としてはよくtanhが用いられる。

そのため、ある時刻tにおける中間層からの出力を、時刻tにおける入力、時刻入力層から中間層への重みとバイアス、中間層から中間層への重みを用いて書くと以下のようになる。

RNNでは各時刻が重みを共有して、中間状態

t时刻の誤差関数 クロスエントロピーと定義される。

在这里，是时刻t上正确的值， 为预测出来的值。使用一整个序列作为一次训练，每个时刻的误差累计构成总的误差。

　この誤差は数値で表現する必要があるため、ネットワーク出力値と正解の値で計算される誤差関数と呼ばれる関数を定義する。結局、誤差を小さくするように重みをチューニングすることは、誤差関数の値が小さくなるように重みをチューニングしていくことになる。誤差関数は、以下のクロスエントロピーなど、問題に応じて何種類かの代表的な関数を使うことが多い

だから、時刻tにおける誤差をとすると、合計誤差は次のように表すことができる。

これから、誤差関数の値E（w） が最小になるような 重みw を求めればよい

すべての重み、、、についてそれぞれの勾配を計算する

まずは、連鎖律を用いると、中間層から出力層への重みに関する誤差関数の偏微分は以下のようになる。

次は、中間層から中間層への重みについての勾配を計算すると以下のようになる。

はｔ時刻の中間層の出力を表すため、式（１）によると、はの関数であり、に関する計算はとに依存する。また前の時刻t-1のときに、であり、 に関する計算はとに依存する。だからを展開する必要がある。

を展開すると

そして、を展開すると、

式（８）を式（7）に代入すると

これにより、式(6)のという部分が展開された。

式（9）を式（6）に代入すると

最後に、入力層から中間層への重みに関する誤差関数の偏微分を計算する。の計算と同様に、連鎖律を用いると、重みに関する誤差関数の偏微分は以下のようになる

を展開すると、

-

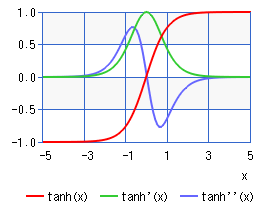
2.2 RNNの勾配に関する問題点

先ほど計算した勾配の特性上、普通のRNNは長時間のデータを意識した学習をすることができないという問題が起こる。

重みに関する誤差関数の偏微分との計算に関しては、時刻tだけではなくて、時刻kからtまで、すべての時刻を一緒に計算する必要があるから、また、総乗の部分が時間に依存するため、時刻ｊと時刻ｋの間が大きく離れた場合は勾配爆発と勾配消失の問題が発生する。

なぜなら、の計算結果は

tanh関数の導関数の値の範囲がはであるため



Tanh関数のグラフ

よって：

勾配爆発が発生するの原因は

>4のとき、 、もしJとKの間の距離が大きすぎると、連乗も多すぎて，勾配爆発が発生し，無限に近づいてしまう。

勾配消失が発生する原因は

<4のとき、であるため、距离过大，会导致连乘项过多，产生梯度消失，趋近于0

勾配消失が起こると、ある程度先の時刻以降の誤差は全く考慮しないで重みを更新するという現象が起こっている。実際には4－５時刻程先の誤差を考慮した勾配を計算するとほぼ0になる。

この勾配消失の問題に対する対策として、簡単な方法としては活性化関数としてReLUを用いることである。

しかし、ReLUは正の領域での勾配が常に１になるので勾配消失が起こりにくくなる。しかし、マイナスの値を全て0にしてしまうので長期時刻順伝播させる際に値がほとんど0になってしまう可能性がある。

RNNもう1つの問題点として、入力層―中間層、中間層―中間層、中間層―出力層で常に共通の重みを用いるために、重要な入力を通すために重みを大きくするように学習が進んでしまうと逆に時系列上にある不必要な情報も大きく通すようになっていまい、

そのため、RNNの重みは学習の際に常に矛盾したアップデートをされていることにより、学習がなかなか進まない。

これらの問題を、時間展開される中間層の構造を変化させることにより回避したのがLSTM（Long Short Term Memory）である。

2.3 改良したRNN―LSTM

LSTM（Long Short Term Memory）的定义和结构

LSTMは、神経科学の短期記憶、長期記憶からヒントを得てデザインされたものである。短期記憶とは、初めて聞いた電話番号を記憶するような場合で、数分程度の一時的な想起は可能であるが、数日程度の長い期間がたつと忘れてしまうような種類の記憶である。

　一方、長期記憶は、自宅の電話番号のようなものであり、こちらは何年もの期間にわたって記憶されている。LSTMは、RNNの中間層出力に対して、記憶期間の長さの考え方を導入することにより、遠い過去の出力の影響を保持することを可能にしている。

LSTMはHochreiter&Schmidhuber在1997年提出的，目前在NLP领域已经取得了很好的应用效果，远远优于其他算法。

従来のRNNの中間層出力は前の時刻の出力と今の入力の重み付き線形和に活性化関数を作用されるだけの形だったため、時系列が長くなればなるほど活性化関数にネストされて、勾配消失が起こる。

LSTMは勾配爆発問題と勾配消失問題をCEC（Constant　Error　Carousel）とという概念を導入するにより解決することができる。

CECは過去の情報を保存するためのユニットである。例えば、前の時刻t-1の出力と今の入力の重み付き線形和をそのまま通すではなく、前の時刻t-1のCECの値を加算して出力される。

LSTM为什么解决了梯度消失，数学分析。

LSTMが勾配消失問題を解決する原因

LSTMを同じように勾配をBPTTで展開すれば、

为了便于分析，如果考虑bias，同时忽略输入变量ht-1的作用，那么隐含层之间的关系可以表示为：

于是，需要连乘的项可表示为：

该值范围在0~1之间，但是在实际参数更新中，可以通过控制bias比较大，使得该值接近于1；在这种情况下，即使通过很多次连乘的操作，梯度也不会消失，仍然可以保留"长距"连乘项的存在。即总可以通过选择合适的参数，在不发生梯度爆炸的情况下，找到合理的梯度方向来更新参数，而且这个方向可以充分地考虑远距离的隐含层信息的传播影响。

另外需要强调的是，LSTM除了在结构上天然地克服了梯度消失的问题，更重要的是具有更多的参数来控制模型；通过四倍于RNN的参数量，可以更加精细地预测时间序列变量。

CECを導入することにより、過去の入力情報をすべて記憶し、過去を遡っても誤差を逆伝播できるようになったが、時系列データを学習するうえで、もう1つ大きな問題がある。時系列データを入力で受け取る場合は、時間依存性がある信号を受け取ったは重みを大きくし活性化すべき一方、依存性がない信号を受け取ったときは重みを小さくし非活性のままであるべきである。しかし実際は、ニューロンが同じ重みでつながっている限り、両者はお互いに打ち消し合う重みの更新となってしまうので、特に長期依存性の学習がうまくできないことに繋がる。

この問題を解決するためには、依存性のある信号を受け取ったときのみ活性化し、それ以外では依存性のありそうな情報を内部で保持しておく機構が必要になる。後者はCECにより実現できるが、前者の問題は、必要になったタイミングでのみ信号を伝播し、それ以外では信号を遮断するゲートのような存在を取り入れるべきである。

ゲートに関しては入力ゲート、忘却ゲートと出力ゲートという3つのゲートの導入することによって各時刻ごとに入出力を制御することが可能になる。入力ゲートと出力ゲートに関しては、CECの入力部分に入力ゲート、出力部分に出力ゲートを導入することで、入出力ともに過去の情報が必要になったタイミングでのみゲートを開け信号を伝播し、それ以外はゲートを閉じておくことで過去の情報を保持しておくことが可能になる。また、忘却ゲートに関してはCECからの誤差を受け取ることで、必要なタイミングでCECに記憶された値を忘れ去る機能を果す。

しかし、ゲートの制御に用いるのは時刻ｔにおける入力層の値がおよび時刻ｔ－１における中間層の値であり、制御すべきCEC自身が保持している値は用いられていない。一見するとを制御に用いていることによりCECの状態が反映されているように思われるが、あくまでもLSTMブロックの出力は出力ゲートに依存しているので、仮に出力ゲートがずっと閉じている場合、どのゲートもCECにアクセスすることができず、CECの状態を見ることができないという問題が発生する。

この問題を解決するために導入されたのが覗き穴結合（peephole connection）である。これはCECから各ゲートをつなぐもので、これによりCECの状態を各ゲートに伝えることができるようになる。

ここまで

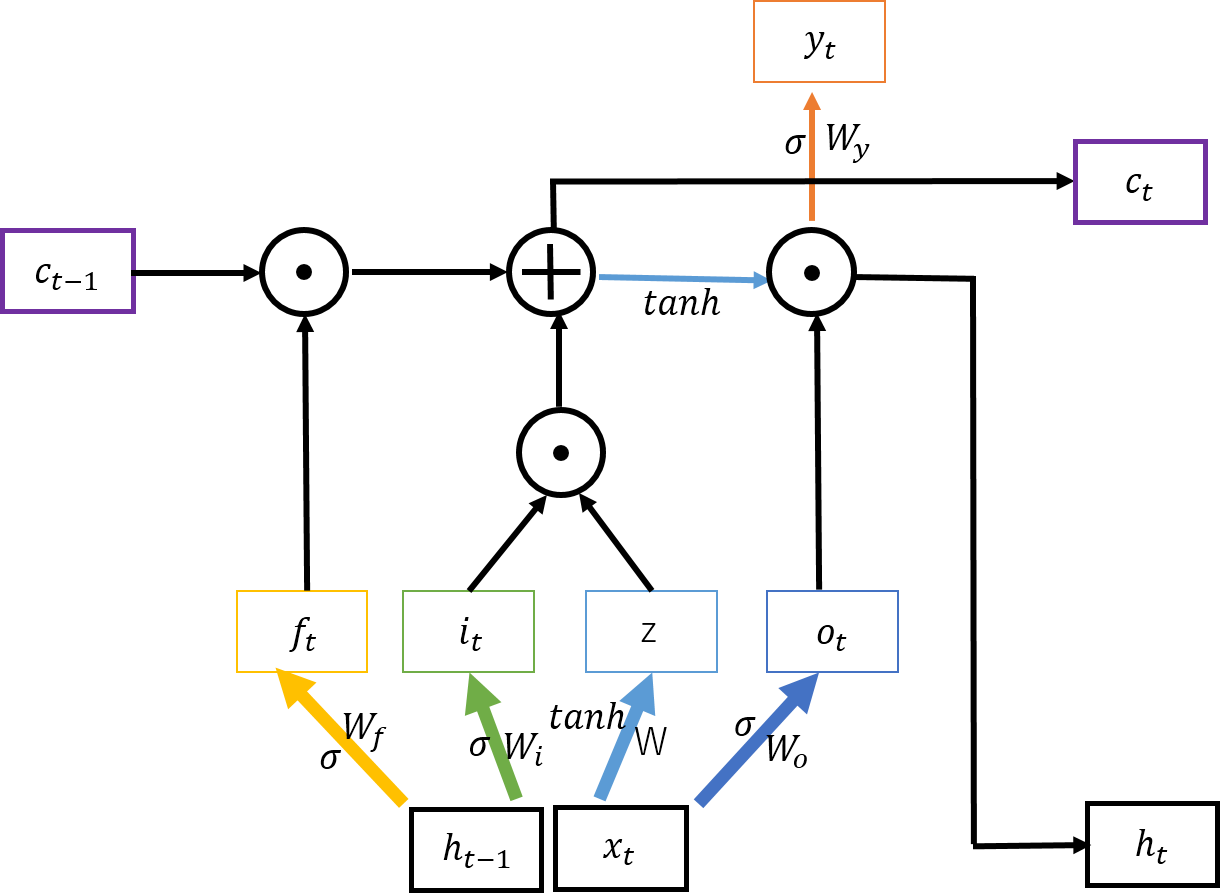
具体的にそれぞれのゲートの式は以下のようになる。加上ct-1

忘却ゲート

入力ゲート

入力データ

出力ゲート



⊙はアダマール積であり、

、、の計算に関してはそれぞれt時刻における入力と前の時刻における中間層の出力の重み付き線形和にシグモイド関数を作用され、0.0～1.0の間の値に調整された。ゲートの値が1.0に近い場合はそれぞれの入力を通過させ、0.0.に近い場合はシャットダウンする。

また、はtanh関数を作用されることにより－1.0～1.0の間の値になるように調整された。はただの入力データであり、ゲートを制御するわけではない。

これから、忘却ゲートを作用させた前の時刻のCECの値Cｔ－１と入力ゲートを作用させた入力を時刻ｔの入力を加算してCECを更新する

それに活性化関数TANHを作用させた上で、出力ゲートを作用させて中間層出力とする。

⊙tanh()

最後の結果の出力は

=

これがLSTMの仕組みである。

、、、を定義する。すると、これらを

という形でまと　。こうしておくことで、を求めればよいことになるので、

よって：

これらの式にある、、、は、

一方、により

3データの前処理

データの中における最大値と最小値を使って正規化する方法です。

この処理をすることで、データは最大値が1,最小値が0のデータとなります。あらかじめ最大値と最小値の範囲が限られている場合には有効な手法です。(逆に、外れ値が存在すると他の値の差がほとんど無くなってしまう可能性があります)

では、これもNumPyで実装していきます。

1999 年に Hochreiter らによって提唱された LSTM-RNNに基づいている。計算するにあたって、学習時には、Epoch 数を 50、100、500 の３つのパターンで計算。隠れ層のユニット数は 50、LSTM 層は 1 とした。学習のフレームワークとしては、フロントエンドに Keras、バックエンドには Tensorflowを使用して実装している。確率的勾配降下法の最適化を行う上で、Adam(Adaptive Moment Estimation) により学習率の設定を行っている。

