RNN

2017年6月23日

今回の内容

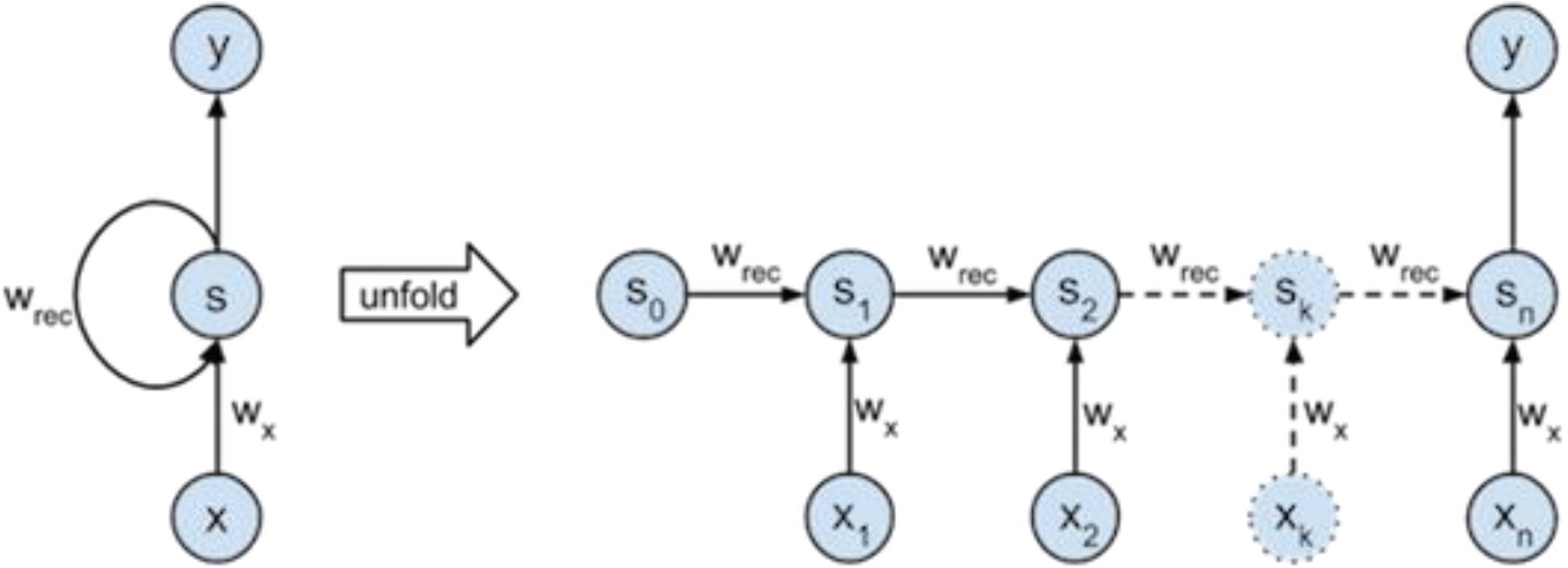
- 2年生向けに、2016秋プロジェクトの最終発表プログラムのデモンストレーション。
- RNN解説。
- RNN一例。
- 輪読(5章)。

RNN

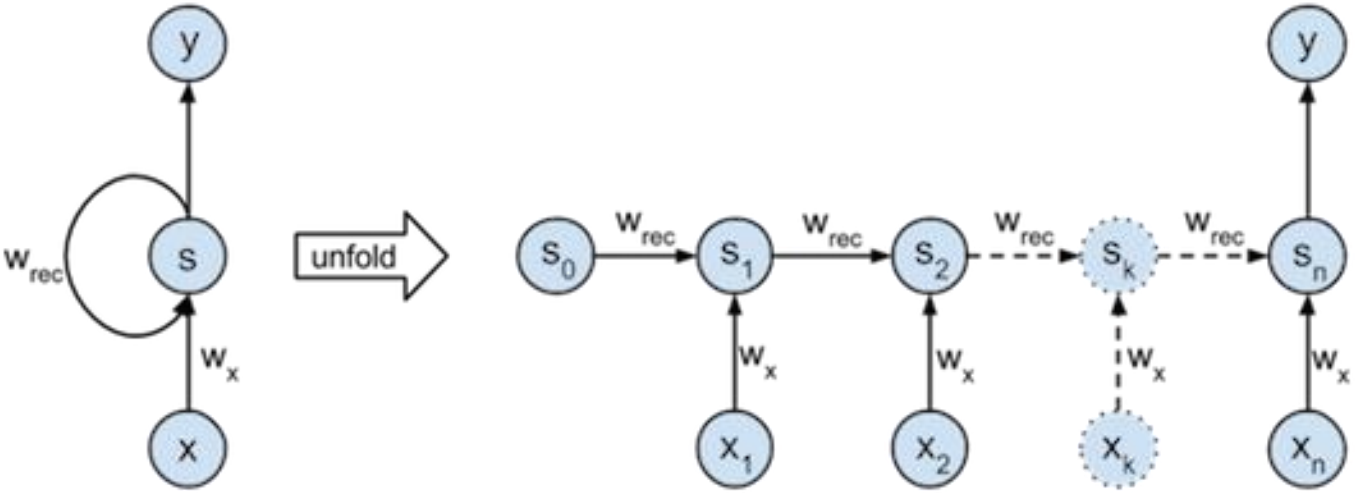
- http://qiita.com/icoxfog417/items/2791ee878deee0d0fd9c
- http://qiita.com/yukiB/items/f6314d2861fc8d9b739f
- 解説・サンプルコードは、上記のページを参考にする。
  コードは、githubに公開されている。
  https://github.com/yukiB/rnntest
  （rnn.py）
- 言語モデルではなく、より単純なモデルを扱う問題を例に挙げ、TensorFlowでのRNN実装を試している。

RNN

- http://peterroelants.github.io/
  から図を引用。ページも参考になる。

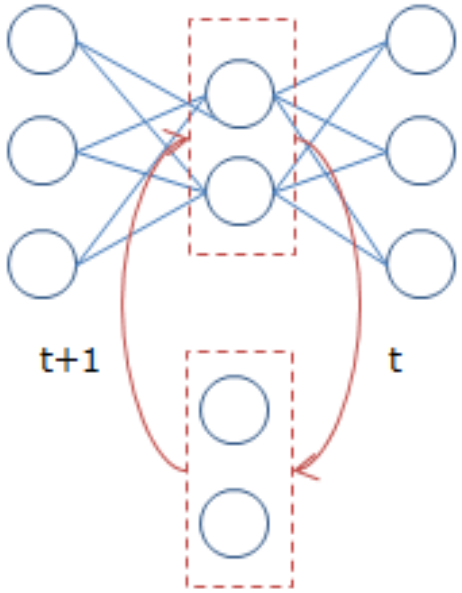


RNNの意味合い



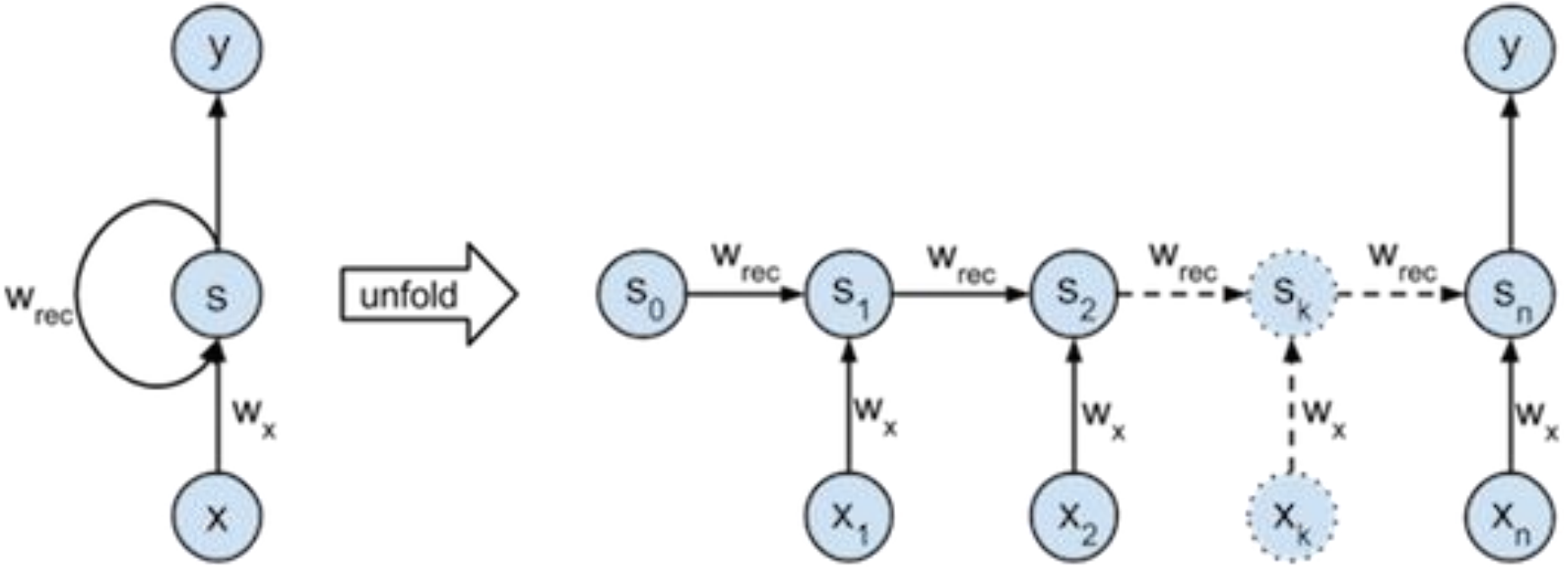
- データの中には、「x」が出たら「y」が来る可能性が高い、というように前のデータが次のデータに対し相関を持つものがあります。具体的には、言葉や音楽といったものです(「私」の後には「は」か「が」が来ることが多い、など)。こうした時系列で相関を持つデータでは、当然前に発生したデータを考慮したくなります。NNに、前に発生したデータを投入できるようにはできないか。その答えが、RNNとなります。

RNNの意味合い



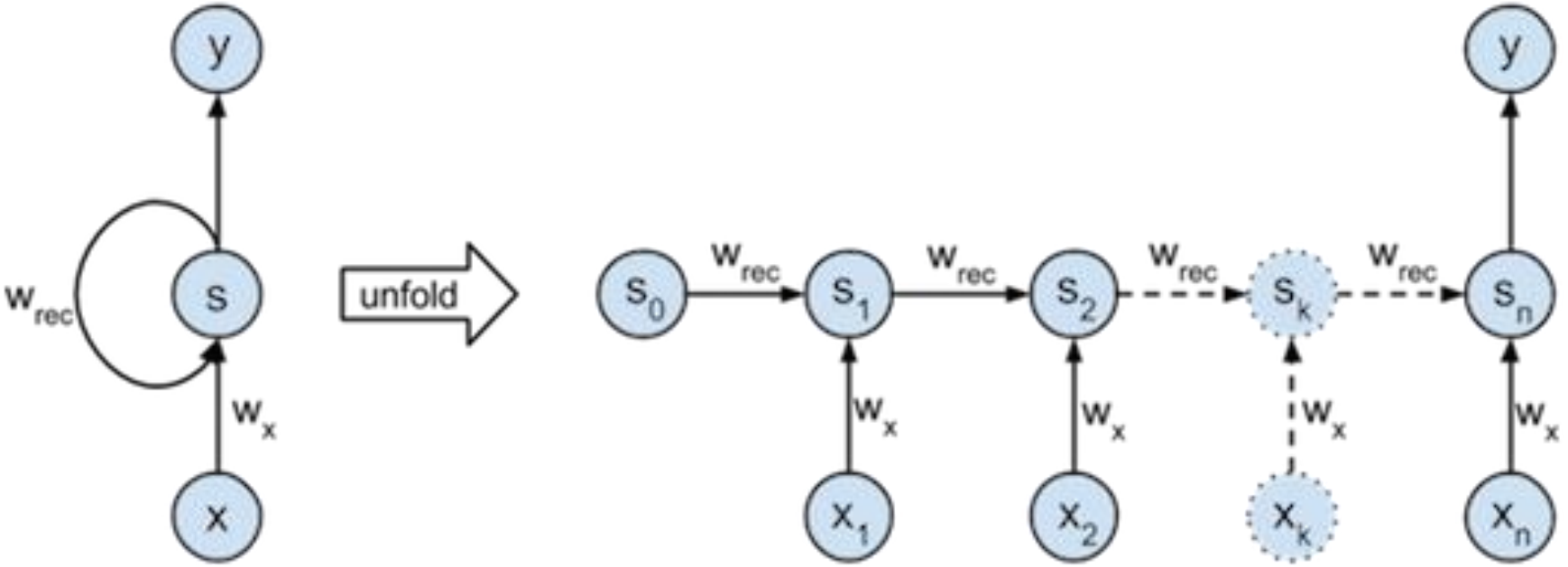
- 時刻tの隠れ層の内容が、次の時刻t+1の時の入力として扱われます。t+1の隠れ層がt+2の・・・と続くわけですが、要は前回の隠れ層が次の隠れ層の学習にも使用される、というイメージです。

SimpleなRNN



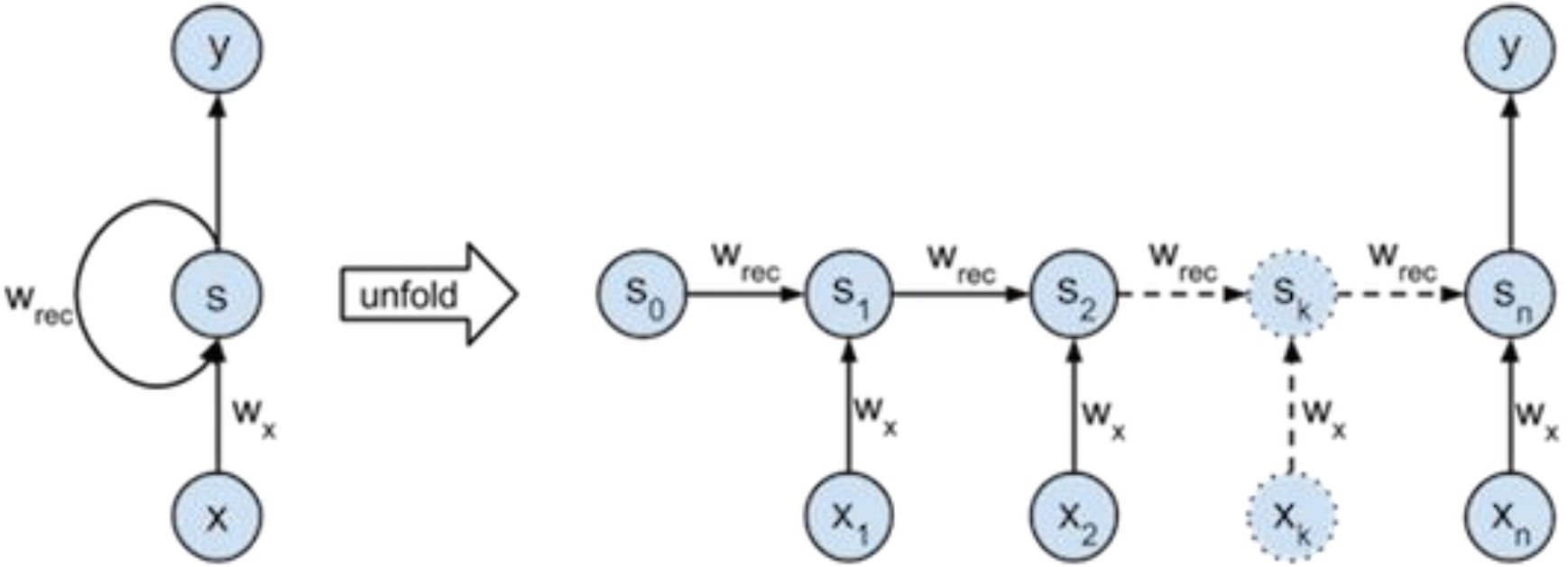
- 入力層ユニットxからのデータに重みW_xを乗じた後，隠れ層ユニットsに入る，ユニットsの出力に対しては再帰がかかり，重みW_recをかけた結果が次のステップのユニットsに入る.

SimpleなRNN



- 上図右の展開された状態を考えると，隠れ層ユニットの初期値s_0の状態は，ステップが進行するにつれ重みW_recを乗じながら状態を変異させていく.

SimpleなRNN



- その際，各ステップにおいてxが入力され，最終ステップのs_nの状態が出力層ユニットyに出力される．

RNNの種別

名称	結合対象	特徴
Fully recurrent network	全ノード(1:N)	自身も含め完全双方向に結合する
Hopfield network	全ノード(1:N-1)	双方向結合で、結合対象に自身は含まない
Elman network	1:1 (隠れ層->隠れ層)	入力層・コンテキスト(隠れ層)・出力層の3層構造
Jordan network	1:1 (出力層->隠れ層)	入力層・コンテキスト(隠れ層)・出力層の3層構造
Echo state network (ESN)	1->1?	結合対象は、ノードの集合(reservoir)からランダムに決定される
Long short term memory network (LSTM)	-	RNNのノードの代わりに、入力値を保持しておけるBlockを採用したもの。高精度
Bi-directional RNN (BRNN)	-	双方向(過去->未来/未来->過去)のRNNを組み合わせたもの

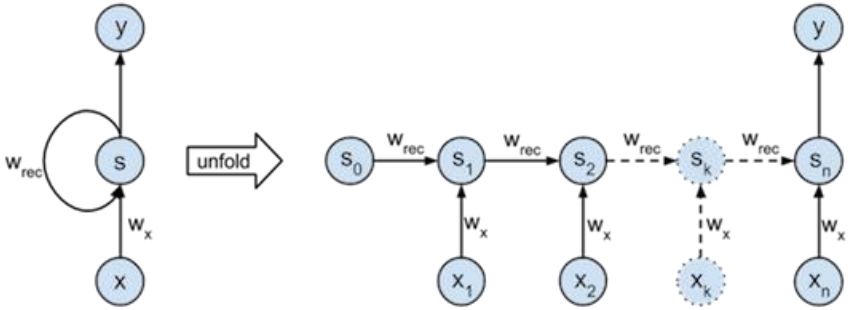
- それぞれの詳細は、http://qiita.com/icoxfog417/items/2791ee878deee0d0fd9c を参照。

今回使用するのはLSTM。

LSTM(Long short term memory network)

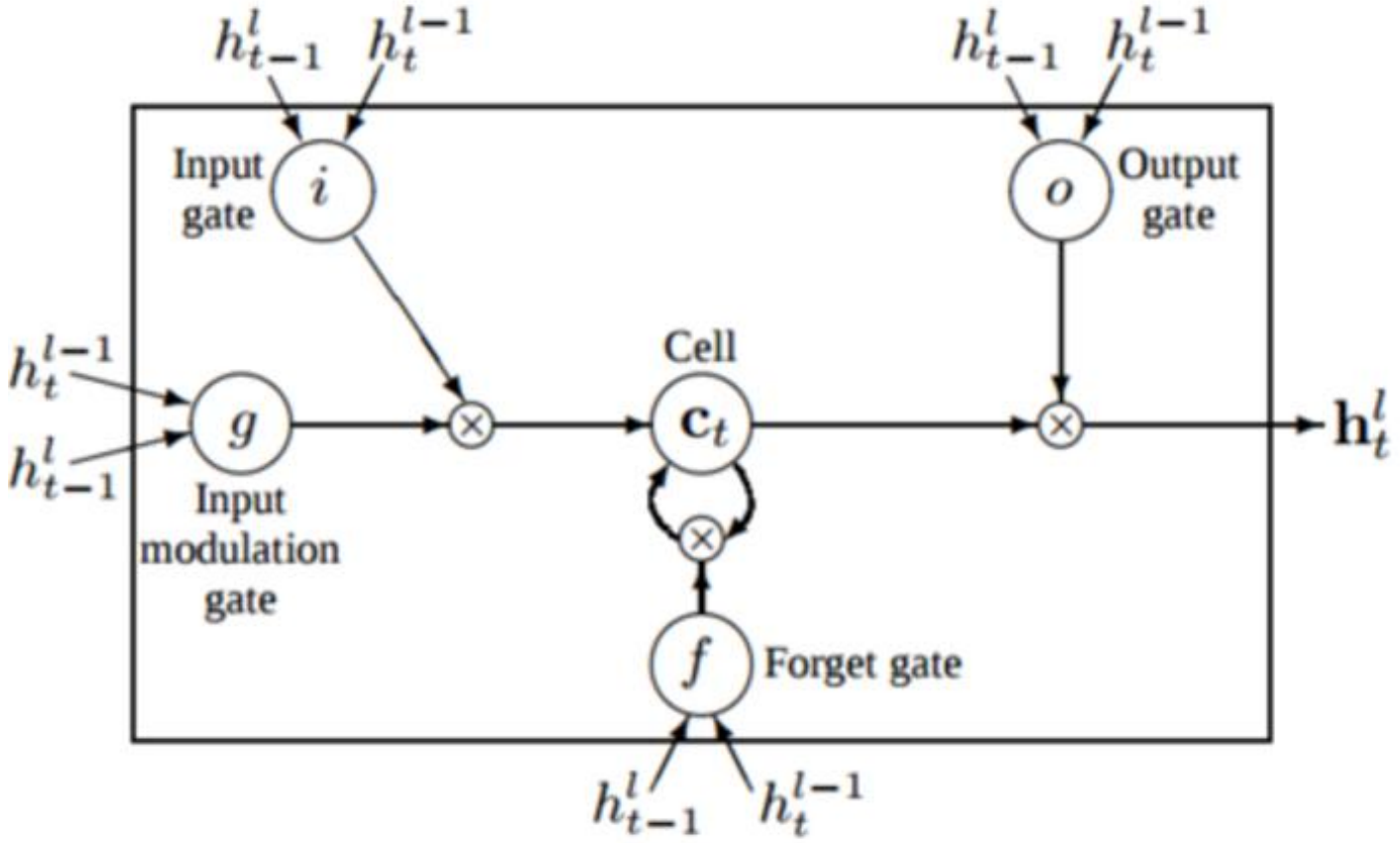
- Tがあまりに大きい、つまり長い時系列のデータの場合、計算上の問題で上層からの誤差が薄まったり逆に非常に大きくなったりします。
  https://www.slideshare.net/beam2d/pfi-seminar-20141030rnn
  参照。
- 値が大きくなる分には最大値の制限で何とかなりますが、消えてしまうのはどうにもならないため、誤差が減衰しないよう伝播させるというのがLSTMの思想です。
- http://colah.github.io/posts/2015-08-Understanding-LSTMs/
  も参照。

LSTMの使用目的



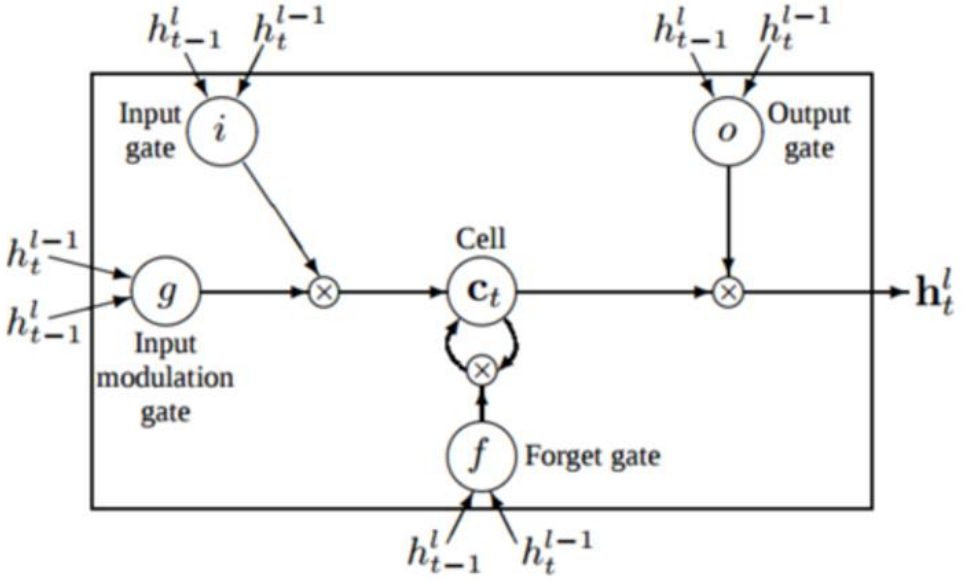
- TensorflowのRNNチュートリアルは、シンプルなRNNではないLSTMで作成されている。
- 前ページの図で示した通り，通常のRNNではNNの大きさがステップ数に比例してどんどん大きくなってしまいます.
- そのため，誤差逆伝搬法を適応させるために必要な計算量やメモリが多くなり，伝搬される誤差が爆発してしまって計算が安定しないという問題がありました.
- それに対しLSTMでは，単純な隠れ層の代わりにLSTMユニットを用いることで，ユニットのコア値（メモリセル値）が次の時刻でどれくらい保持されるか，そして次のステップにどれだけ影響するかを調整することができます.

LSTMユニット



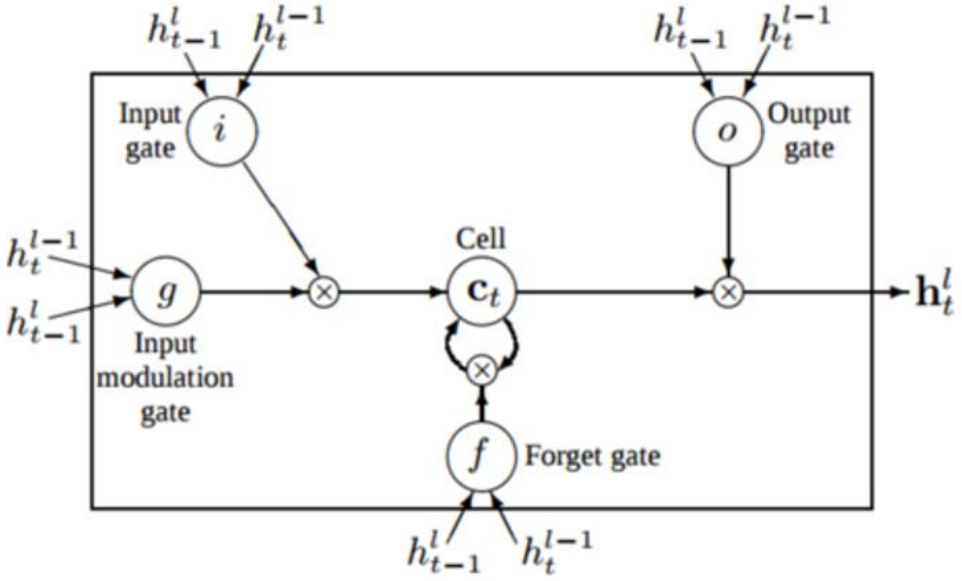
LSTMユニット一つが上図のようになっている。

LSTMユニット



- メモリセル（Cell）・・・過去の状態を表し，C_t で表す．
- 入力 (input gate)・・・時刻tにおけるl–1番目の隠れ層の出力 h_t^{l-1} と時刻t–1におけるl番目の隠れ層の出力 h_{t-1}^{l} からなる．
- 入力判断ゲート (input modulation gate)・・・メモリセルに加算される値を調整する役割を持つ

LSTMユニット



- 忘却判断ゲート (forget gate)・・・メモリセルの値が次の時刻でどれくらい保持されるかを調整する役割を持つ
- 出力判断ゲート (output gate)・・・メモリセルの値が次の層にどれだけ影響するかを調整する役割を持つ

RNNサンプルモデル

- 参考にしたモデルの出力結果の一部は以下のようになる。

```
[1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0]
output: 8.004877, correct: 8
```

- 今回のモデルは、1000個の0と1のみで表現されたtrainデータ(リスト)を学習した後、100個の同様に表現したtestデータの中の数値の合計を出力できるか目指すモデルとなっている。上図では、正解8に対して8.004877と正しく出力できたといえる(誤差0.05未満を正解としている)。
100個全てのtestデータで出力を出し、正答率を最終的に算出している。
- 「数値をカウントするというアルゴリズム」は使わずにRNN(含む2つの重み係数）で推定する

rnn.py

- Tensorflowにデフォルトで実装されているBasicLSTMCellを使って作成されている。(https://github.com/yukiB/rnntest)
  しかし、そのまま実行するとprintの部分でエラーになってしまうため、修正が必要となる。(print_result関数)
- 修正例:

```
92        def print_result(i, p, q):
93            [print(list(x)[0]) for x in i]
94            print("output: %f, correct: %d" % (p, q))
```

```
def print_result(i,p, q):
    l=[]
    for j in range(len(i)):
        l.append(float(i[j]))
    print (l)
    print("output: %f, correct: %d" % (p , q))
```

rnn.py実行

- 学習100回毎に誤差、500回毎に正答率を表示している（左）。
- 指定回数学習が終わると、100個それぞれの詳細も表示(右)。

```
train#0, train loss: 2.294682e+01
accuracy 0.000000
train#100, train loss: 8.663831e-01
train#200, train loss: 7.782108e-02
train#300, train loss: 1.349167e-01
train#400, train loss: 7.798133e-02
train#500, train loss: 1.288390e-01
accuracy 0.000000
train#600, train loss: 7.182720e-02
train#700, train loss: 4.385892e-02
train#800, train loss: 7.584190e-02
train#900, train loss: 2.229058e-02
train#1000, train loss: 2.854447e-02
accuracy 0.060000
train#1100, train loss: 4.982641e-02
train#1200, train loss: 2.626819e-02
train#1300, train loss: 4.285531e-03
train#1400, train loss: 5.666641e-02
```

```
[0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 0.0]
output: 2.973547, correct: 3
[1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0]
output: 8.004877, correct: 8
[1.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0, 1.0]
output: 3.988324, correct: 4
[1.0, 0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0]
output: 4.983786, correct: 5
[0.0, 1.0, 0.0, 1.0, 1.0, 1.0, 0.0, 1.0, 1.0, 0.0]
output: 5.969229, correct: 6
[1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0]
output: 8.024482, correct: 8
[0.0, 0.0, 0.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 0.0]
output: 6.013995, correct: 6
[0.0, 1.0, 1.0, 1.0, 1.0, 0.0, 1.0, 0.0, 1.0, 0.0]
output: 5.977817, correct: 6
[0.0, 1.0, 0.0, 1.0, 0.0, 1.0, 1.0, 0.0, 1.0, 1.0]
output: 5.975311, correct: 6
accuracy 0.970000
```

RNNサンプルコード(各種パラメータ)

```
num_of_input_nodes = 1
num_of_hidden_nodes = 80
num_of_output_nodes = 1
length_of_sequences = 10
num_of_training_epochs = 1000
size_of_mini_batch = 100
num_of_prediction_epochs = 100
learning_rate = 0.01
forget_bias = 0.8
num_of_sample = 1000
```

RNNサンプルコード(各種パラメータ)

- num_of_input_nodes: 入力層ノード数
- num_of_hidden_nodes: 隠れ層ノード数
- num_of_output_nodes: 出力層ノード数
- length_of_sequences: train,testデータの次元数
- num_of_training_epochs: (意味合いとしては)学習回数
- num_of_prediction_epochs: (意味合いとしては)testデータの個数
- learning_rate: 学習率
- forget_bias: 訓練開始時の忘却の規模を減らす(本来のデフォルトは1.0)
- num_of_sample: trainデータの個数

RNNサンプルコード

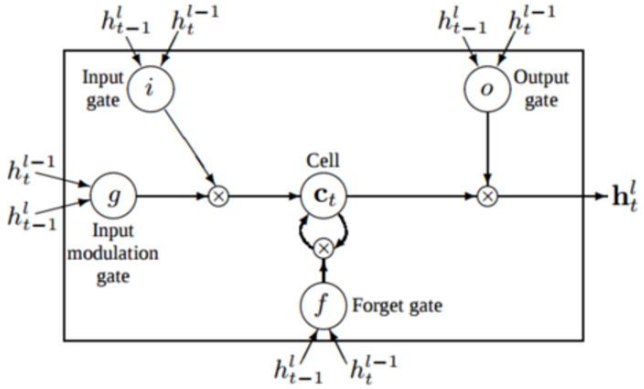
- データの作成。サンプルでは、train用に1000個、test用に100個のデータを生成することになる。（下図はtest用）

```
def create_data(nb_of_samples, sequence_len):
    X = np.zeros((nb_of_samples, sequence_len))
    for row_idx in range(nb_of_samples):
        X[row_idx, :] = np.around(np.random.rand(sequence_len)).astype(int)
    # Create the targets for each sequence
    t = np.sum(X, axis=1)
    return X, t
```

```
X:[[ 1.  0.  0.  0.  0.  1.  0.  0.  0.  1.]
 [ 1.  0.  1.  0.  1.  0.  0.  0.  1.  0.]
 [ 1.  1.  0.  1.  1.  1.  1.  0.  0.  0.]
 [ 1.  0.  1.  1.  0.  1.  0.  0.  1.  1.]
 [ 1.  0.  1.  1.  1.  0.  0.  1.  0.  0.]
 [ 0.  0.  1.  0.  1.  1.  0.  0.  0.  1.]
 [ 1.  1.  1.  1.  0.  1.  0.  1.  0.  1.]
 [ 0.  0.  0.  1.  0.  0.  1.  1.  0.  0.]
 [ 0.  0.  0.  0.  1.  0.  1.  0.  1.  1.]
 [ 0.  1.  0.  0.  1.  0.  1.  0.  0.  1.]
 [ 1.  1.  1.  0.  1.  0.  1.  0.  0.  0.]
 [ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
```

```
t:[ 3.  4.  6.  6.  5.  4.  7.  3.  4.  4.  5.  1.  7.  6.  5.  4.  4.  4.
  6.  4.  4.  5.  3.  8.  3.  5.  5.  4.  3.  2.  5.  5.  7.  6.  3.  4.
  3.  6.  4.  1.  8.  5.  4.  5.  6.  4.  4.  5.  6.  7.  7.  3.  8.  4.
  4.  5.  3.  7.  5.  7.  7.  5.  6.  3.  4.  6.  4.  5.  5.  3.  1.  4.
  6.  3.  3.  2.  4.  5.  7.  5.  7.  2.  8.  7.  6.  6.  4.  5.  1.  5.
  4.  5.  4.  3.  5.  7.  3.  9.  4.  3.]
```

RNNサンプルコード(LSTM層)



- LSTM層の設計。

```
def inference(input_ph, istate_ph):
    with tf.name_scope("inference") as scope:
        weight1_var = tf.Variable(tf.truncated_normal(
            [num_of_input_nodes, num_of_hidden_nodes], stddev=0.1), name="weight1")
        weight2_var = tf.Variable(tf.truncated_normal(
            [num_of_hidden_nodes, num_of_output_nodes], stddev=0.1), name="weight2")
        bias1_var = tf.Variable(tf.truncated_normal([num_of_hidden_nodes], stddev=0.1), name="bias1")
        bias2_var = tf.Variable(tf.truncated_normal([num_of_output_nodes], stddev=0.1), name="bias2")

        in1 = tf.transpose(input_ph, [1, 0, 2])
        in2 = tf.reshape(in1, [-1, num_of_input_nodes])
        in3 = tf.matmul(in2, weight1_var) + bias1_var
        in4 = tf.split(0, length_of_sequences, in3)

        cell = tf.nn.rnn_cell.BasicLSTMCell(
            num_of_hidden_nodes, forget_bias=forget_bias, state_is_tuple=False)
        rnn_output, states_op = tf.nn.rnn(cell, in4, initial_state=istate_ph)
        output_op = tf.matmul(rnn_output[-1], weight2_var) + bias2_var

        # Add summary ops to collect data
        w1_hist = tf.histogram_summary("weights1", weight1_var)
        w2_hist = tf.histogram_summary("weights2", weight2_var)
        b1_hist = tf.histogram_summary("biases1", bias1_var)
        b2_hist = tf.histogram_summary("biases2", bias2_var)
        output_hist = tf.histogram_summary("output",  output_op)
        results = [weight1_var, weight2_var, bias1_var,  bias2_var]
        return output_op, states_op, results
```

RNNサンプルコード(誤差計算)

- コストの計算については，今回出力値が連続の値をとる数値のため，MSE(mean square error:平均二乗誤差)が適切。

```
def loss(output_op, supervisor_ph):
    with tf.name_scope("loss") as scope:
        square_error = tf.reduce_mean(tf.square(output_op - supervisor_ph))
        loss_op = square_error
        tf.scalar_summary("loss", loss_op)
        return loss_op
```

$$\mathrm{RMS}[x] = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i}$$

RNNサンプルコード(正答率計算)

- testデータを設定数作成し、それぞれ予測結果と正解の差が0.05未満のものの割合を求めている．absは絶対値
  差を求める例(右図):8.004877-8＝…

```
[1.0, 1.0, 1.0, 1.0, 1.0, 0.0, 0.0, 1.0, 1.0, 1.0]
output: 8.004877, correct: 8
```

```
def calc_accuracy(output_op, prints=False):
        inputs, ts = make_prediction(num_of_prediction_epochs)
        pred_dict = {
                input_ph:  inputs,
                supervisor_ph: ts,
                istate_ph:    np.zeros((num_of_prediction_epochs, num_of_hidden_nodes * 2)),
        }
        output= sess.run([output_op], feed_dict=pred_dict)

        def print_result(i,p, q):
            l=[]
            for j in range(len(i)):
                l.append(float(i[j]))
            print (l)
            print("output: %f, correct: %d" % (p , q))
        if prints:
            [print_result(i,p, q)  for i, p, q in zip(inputs,output[0], ts)]

        opt = abs(output - ts)[0]
        total = sum([1 if x[0] < 0.05 else 0 for x in opt])
        print("accuracy %f" % (total/float(len(ts))))
        return output
```

RNNサンプルコード(ミニバッチ用記述)

- mainの学習部分でミニバッチ学習の記述があるが、その際に指定個数(デフォルトは100)だけランダムに1000個のtrainデータの中から選択する。

```
def get_batch(batch_size, X, t):
    rnum = [random.randint(0, len(X) - 1) for x in range(batch_size)]
    xs = np.array([[[y] for y in list(X[r])] for r in rnum])
    ts = np.array([[t[r]] for r in rnum])
    return xs, ts
```

RNNサンプルコード(main部分、学習準備)

- 赤く囲っているのは、関数の呼び出し部分。
- 学習データ、LSTM層等の学習に使用するものを設定。

```
random.seed(0)
np.random.seed(0)
tf.set_random_seed(0)

optimizer = tf.train.GradientDescentOptimizer(learning_rate=learning_rate)
X, t = create_data(num_of_sample, length_of_sequences)

with tf.Graph().as_default():
    input_ph = tf.placeholder(tf.float32, [None, length_of_sequences, num_of_input_nodes], name="input")
    supervisor_ph = tf.placeholder(tf.float32, [None, num_of_output_nodes], name="supervisor")
    istate_ph = tf.placeholder(tf.float32, [None, num_of_hidden_nodes * 2], name="istate")

    output_op, states_op, datas_op = inference(input_ph, istate_ph)
    loss_op = loss(output_op, supervisor_ph)
    training_op = training(loss_op)

    summary_op = tf.merge_all_summaries()
    init = tf.initialize_all_variables()
```

RNNサンプルコード(main部分、学習)

- 赤く囲っているのは、関数の呼び出し部分。
- ミニバッチ学習で学習する。学習100回毎に誤差、500回毎に正答率を表示している。

```
with tf.Session() as sess:
    saver = tf.train.Saver()
    summary_writer = tf.train.SummaryWriter("/tmp/tensorflow_log", graph=sess.graph)
    sess.run(init)

    for epoch in range(num_of_training_epochs):
        inputs, supervisors = get_batch(size_of_mini_batch, X, t)
        train_dict = {
            input_ph:      inputs,
            supervisor_ph: supervisors,
            istate_ph:     np.zeros((size_of_mini_batch, num_of_hidden_nodes * 2)),
        }
        sess.run(training_op, feed_dict=train_dict)

        if (epoch) % 100 == 0:
            summary_str, train_loss = sess.run([summary_op, loss_op], feed_dict=train_dict)
            print("train#%d, train loss: %e" % (epoch, train_loss))
            summary_writer.add_summary(summary_str, epoch)
            if (epoch) % 500 == 0:
                calc_accuracy(output_op)

    calc_accuracy(output_op, prints=True)
    datas = sess.run(datas_op)
    saver.save(sess, "model.ckpt")
```

RNNをテキストに使用した例

- http://www.madopro.net/entry/char_level_lm_with_simple_rnn
  (Tensorflowで言語モデル、文字レベル)
- http://kiyukuta.github.io/2013/12/09/mlac2013_day9_recurrent_neural_network_language_model.html
  (文章に適用した場合の理論について)
- http://tensorflow.classcat.com/2016/03/21/rnn-2-lang-model-with-numpy-and-theano/
  (GPU使用のTheanoを使ってのテキスト生成まで行っている、かなり本格的)

3年生輪読予定表改訂版

担当箇所	名前	発表予定日
1-1	保科	5/12
1-3	猪狩	5/12
2-1	村田 西澤	5/19
2-2 2-3	西ヶ谷 坂中	5/19
3-1	米倉 田之井	5/26
3-2	松田 清水	5/26

3年生輪読予定表改訂版

担当箇所	名前	発表予定日
3-3	宿野 前原 村田	6/2
4章	門木 山下 上野 數見	6/9 6/16
5章	**下窪** **黒川** **関根** **坂本**	**6/23**