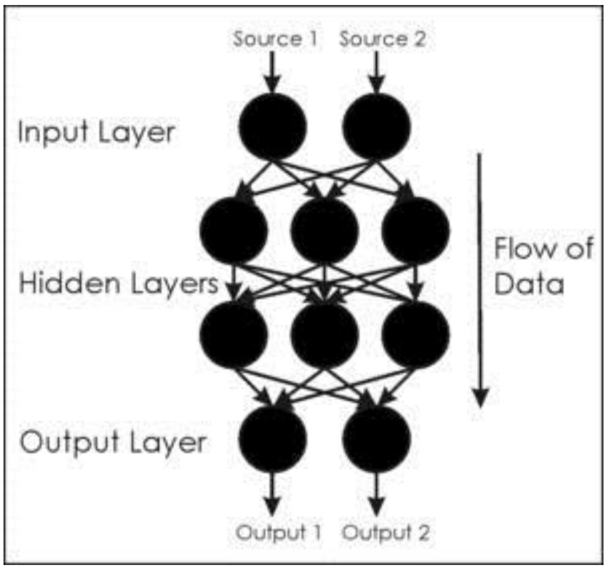
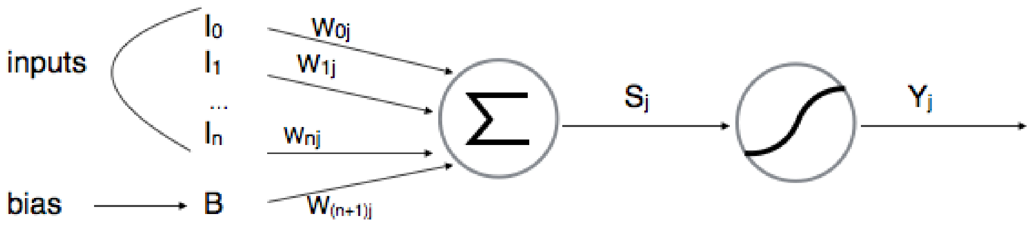
A brief review about how RNN works in tensorflow

Zhengfei 2016.12.17

Outline

General feed-forward neural networks:



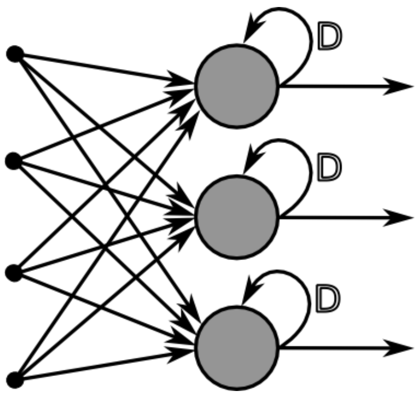


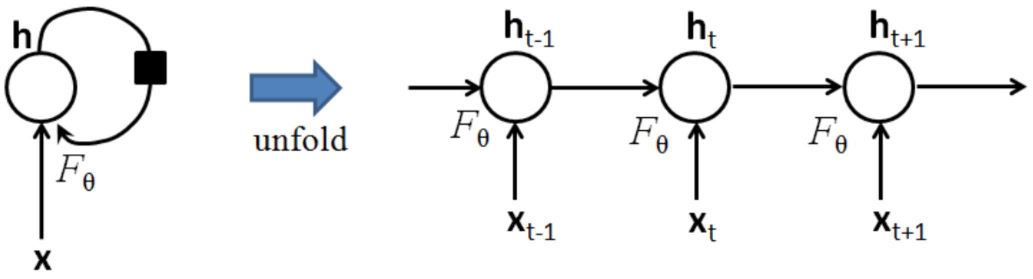
Activation in the neuron: $s_j = \sum_i \omega_{ij} y_i$

Output: $y_j = f_j(s_j)$,

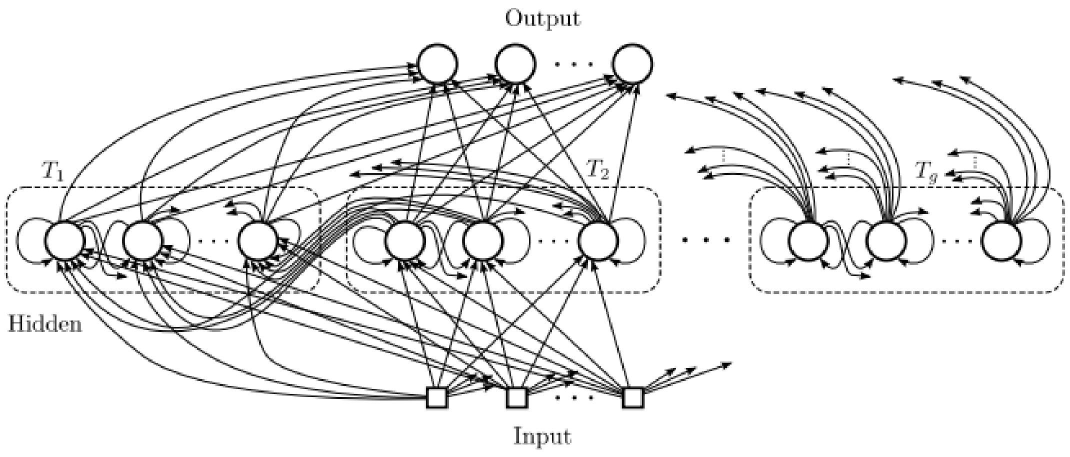
f_j is the activation function, usually is sigmoid, tanh or ReLU

RNN (Recurrent neural network, not to be confused with Recursive neural network.)





More vivid portrayal to descript the process



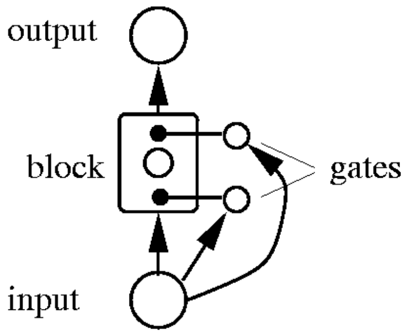
Unlike feedforward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs.

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h)$$
$$y_t = \sigma_y(W_y h_t + b_y)$$

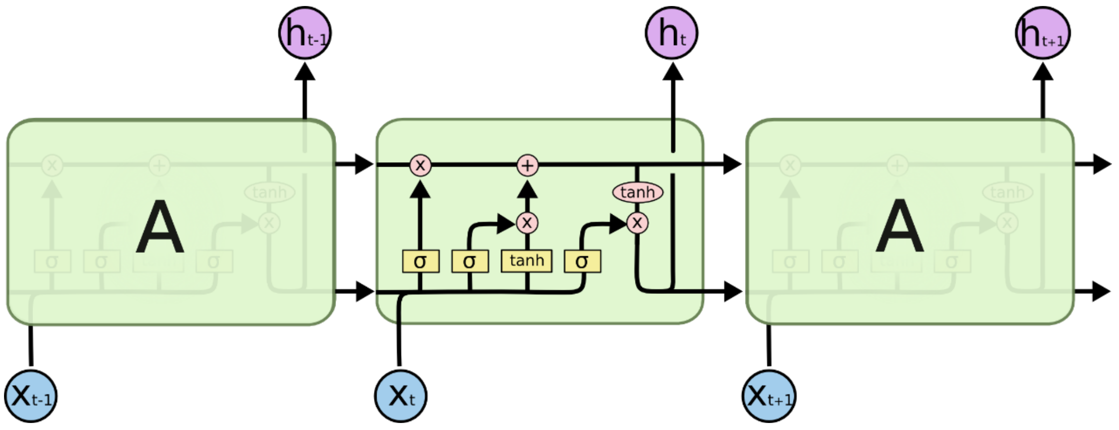
There are very long time lags of unknown size between important events for traditional RNN, and the gradient or blame term tends to vanish when we use the Back propagation algorithm to train the data. So use LSTM model instead of RNN in tensorflow.

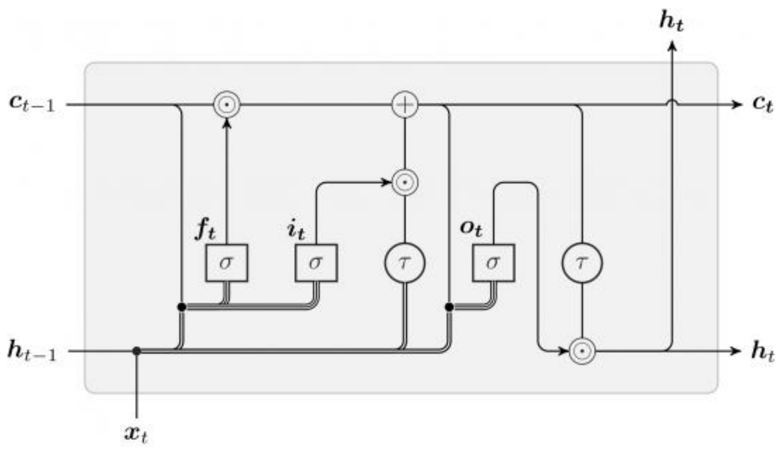
LSTM (http://colah.github.io/posts/2015-08-Understanding-LSTMs/)

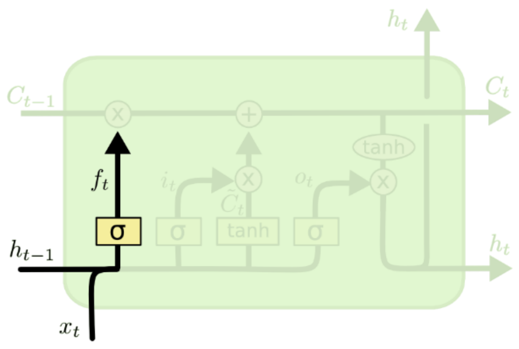
LSTMs are essentially the same thing as the RNN we will develop in this tutorial, they just have a different way of computing the hidden state.



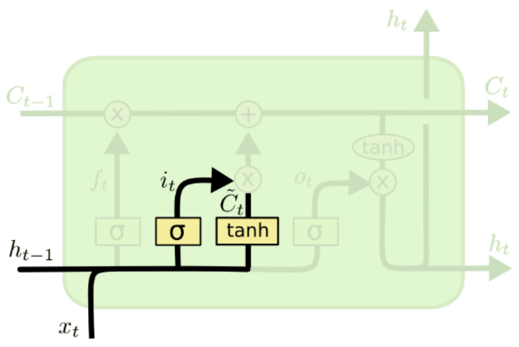
Standard LSTM models



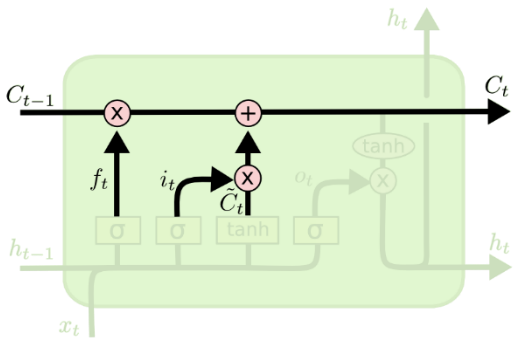




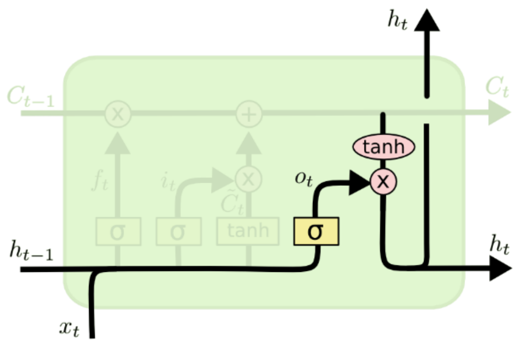
$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$



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



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$



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

Back propagation through time (BPTT, http://ir.hit.edu.cn/~jguo/docs/notes/bptt.pdf)

http://neuralnetworksanddeeplearning.com/chap2.html

We know in tensorflow, it use BPTT algorithm to train the model parameters.

First understand Back propagation

Cost function:

$$C = \frac{1}{2}\sum_j (o_j^l - y_j^l)^2$$

y_j^l : desired output of node *j* in layer *l*;

o_j^l : activation function output of node *j* in layer *l* , $\sigma(x)=\frac{1}{1+e^{-x}},\left(\sigma'=\sigma(1-\sigma)\right)$

x_j^l input to node *j* in layer l

ω_{ij}^l - weight from node *j* in layer *l*-1 to node *k* in layer *l*

b_j^l bias of node *j* in layer *l*

Calculate the gradient: $\frac{\partial C}{\partial \omega_{jk}^l}$, $\frac{\partial C}{\partial b_j^l}$

Back propagation algorithm:

step 1: Run the network forward with input data to get the output

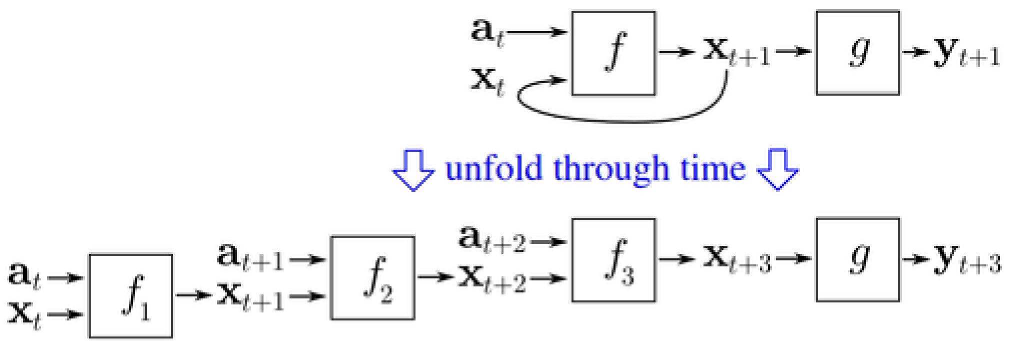
step 2: for each output layer node, compute: $\delta_k = o_k(1-o_k)(o_k - y_k)$

step 3: for each hidden layer node, compute: $\delta_j = o_j(1-o_j)\sum_k o_k\omega_{jk}$

step 4: update the weights and biased as follows: $\begin{matrix} \Delta\omega = -\eta\delta_l o_{l-1}, & \Delta b = \eta\delta_l \\ \Delta\omega + \omega \to \omega, & \Delta b + b \to b \end{matrix}$

η *is the learning rate.*

BPTT change the update error, using time sequence output –input pairs, for example a network has been unfolded to a depth of 3.



Waycare Case study

In tenforflow, in fact, we do not need to care about how it train the data inside.
In our case, the input dimension is length we choose in configuration vector,

configuration vector	No
[	
ac_type	#1
holiday	#2
precipitation	#3
visibility	#4
wind	#5
wind_direction	#6
fog	#7
rain	#8
sun_rise	#9
sun_set	#10
weekend	#11
workday	#12
t0	#13
t1	#14
t2	#15
t3	#16
]	

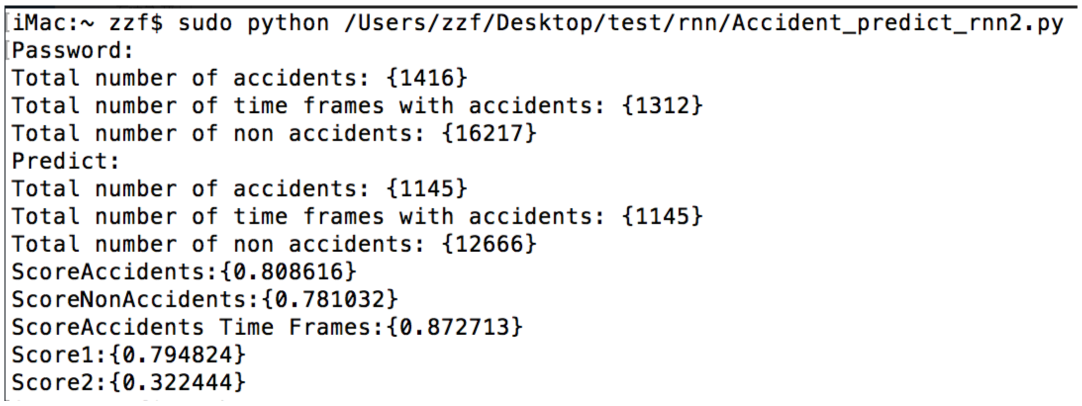
For example in my code , I choose [1,2,3,4,5,7,8,9,10,11,12,13,14,15,16], the input dimension is 15.

Output is only one dimension: Ac_num, the first column

I use only one hidden layer for simplification, in the hidden layer there are four neurons, if we want to use multi layers, we should change the tf function' rnn_cell.BasicLSTMCell' as 'rnn_cell.MultiRNNCell'

The learning rate in the case I choose 0.02.

Dividing one original data set into two parts, one is for training, the other part is for test. For example, I have divided the 2hours.csv into two parts.



```
iMac:~ zzf$ sudo python /Users/zzf/Desktop/test/rnn/Accident_predict_rnn2.py
Password:
Total number of accidents: {1416}
Total number of time frames with accidents: {1312}
Total number of non accidents: {16217}
Predict:
Total number of accidents: {1145}
Total number of time frames with accidents: {1145}
Total number of non accidents: {12666}
ScoreAccidents:{0.808616}
ScoreNonAccidents:{0.781032}
ScoreAccidents Time Frames:{0.872713}
Score1:{0.794824}
Score2:{0.322444}
```

The first part for training, including 4000 rows. The other 16217 rows is for test.

Some Selected References:

RNN and LSTM
http://www.mitpressjournals.org/doi/pdf/10.1162/neco.1997.9.8.1735
https://arxiv.org/pdf/1409.2329v5.pdf
https://arxiv.org/pdf/1402.3511v1.pdf
https://arxiv.org/pdf/1502.04623.pdf
https://arxiv.org/pdf/1303.5778v1.pdf
http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf
http://www.explainthatstuff.com/introduction-to-neural-networks.html
http://jmlr.org/proceedings/papers/v37/jozefowicz15.pdf
http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/

Back-propagation algorithm:
https://www.youtube.com/watch?v=aVId8KMsdUU
https://www.youtube.com/watch?v=zpykfC4VnpM
Algorithms In machine learning online course: https://see.stanford.edu/course/CS229/44
https://web.stanford.edu/class/psych209a/ReadingsByDate/02_25/Williams%20Zipser95RecNets.pdf
https://www.youtube.com/watch?v=bGps-s4_bZE
Vanishing gradient in Back propagation algorithm: http://www.wildml.com/2015/10/recurrent-neural-networks-tutorial-part-3-backpropagation-through-time-and-vanishing-gradients/
http://www.willamette.edu/~gorr/classes/cs449/rtrl.html