Recurrent Neural Network

Astrini Sie

asie@seattleu.edu

Week 9

ECEGR4750 - Introduction to Machine Learning
Seattle University

November 16, 2023

Recap and Updates

- Lab Take Home Assignment due next Wednesday 11/22/23 at 11.59pm
- Office Hours: please email me your desired timeslot and to make appointments
- Final Homework 5 due: 12/1/23 at 11.59pm
- Final week of class: review + in class "exam" on 11/30/23 as bonus
- Final project due: 12/8/23 at 11.59pm
- Please review the notes for this lecture from the original source (a very good lecture and class)

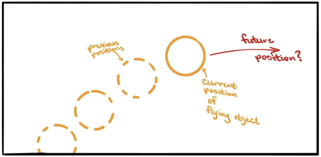
Overview

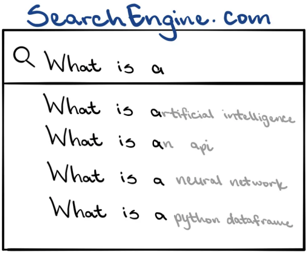
1. Recurrent Neural Network
   - Architecture of an RNN
   - Backpropagation Through Time
   - Limitations

2. Long Short Term Memory (LSTM)

Recurrent Neural Network (RNN)

- Deep learning approach for modeling sequential data (such as time series and language).
- Remembers “history”.
- More modern variants include the Long-Short Term Memory (LSTM) and attention-based models like transformers.





source

Architecture of an RNN

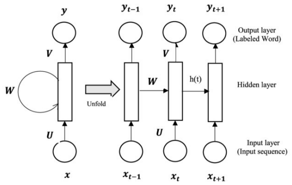
Think of the RNN as a set of singular feed-forward models, where each model is linked together by the internal state update.

Left:

- Simple RNN with input, hidden, and output nodes.
- Input x is sequential with $t+1$ elements.

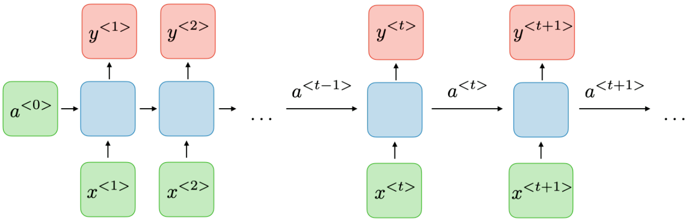
Right:

- "Unrolled" representation of the RNN.
- Each "layer" shares the same structure, weights, and activation functions.
- Working memory for each iteration is passed to the next.



source

Architecture of an RNN



source

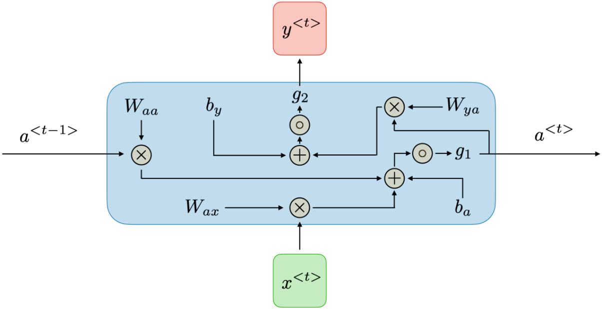
For each timestep t, the activation $a^{<t>}$ and the output $y^{<t>}$ are expressed as:

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$
$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y)$$

where W_{ax}, W_{aa}, W_{ya}, b_a, and b_y are the weights and biases shared temporally; and g_1 and g_2 are the activation functions.

Architecture of an RNN

Zooming in to one hidden node:



source

Architecture of an RNN

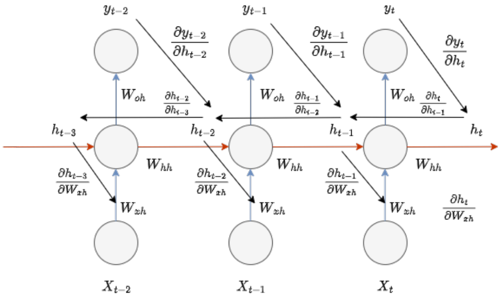
Common activation functions used in RNN:

1. Sigmoid
2. Tanh
3. ReLU

Backpropagation Through Time (BPTT)

1. BPTT works backwards through the chain, calculating the loss and loss gradients across each unrolled ANN in the chain
2. The network is rolled up and the weights are updated

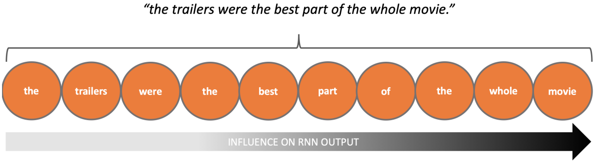
For further reading and derivation, check this out.



source

Limitations of RNN

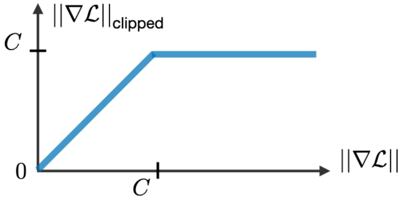
The Vanishing and Exploding Gradient Problems



source

- When the "working memory" of an RNN struggles to retain long term dependencies.
- As the number of hidden layers increases, or as the width of the unrolled RNN increases, BPTT is performed over multiple (long) time steps. Error gradients get amplified or diminished making gradient descent and weight update tricky.

The exploding gradient problem can be solved by **gradient clipping**. The maximum value of the gradient is capped so that it doesn't grow uncontrollably.



source

Limitations of RNN

Solving the Vanishing Gradient Problem

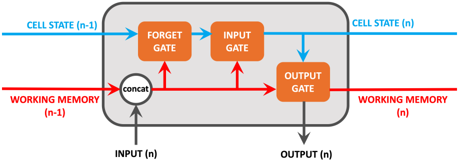
To solve the vanishing gradient problems, variants of the RNN were created: **Gated Recurent Unit (GRU)** and **Long Short Term Memory (LSTM)**.

LSTM is a generalized version of GRU and is a fairly popular solution for time series and early language related machine learning tasks.

Architecture of an LSTM

LSTM introduce new types of mechanisms that help regulate the vanishing gradient problem:

1. **Cell state**. Cell state persists (long-term) information over all iterations of the node. It can be amended to remove or keep information. such that important info from early iterations will not be lost over long sequences.
2. **Forget gate**. Decides information to be removed from the cell state.
3. **Input gate**. Decides information to be added to the cell state.
4. **Output gate**. Decides the working memory this node will output.

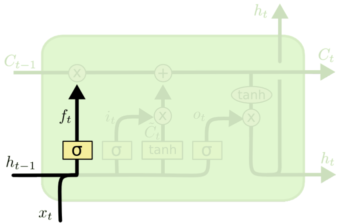


source

Architecture of an LSTM

Forget Gate

What information to throw away from the cell state.
Input: h_{t-1} (hidden state from the previous time step), x_t (input at current time step).
Output: a number between 0 and 1 for each number in the cell state C_{t-1}.

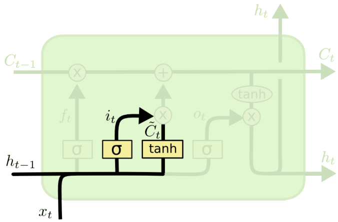


$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \; + \; b_f\right)$$

source

Example: Predicting next word based on a previous ones. If the cell state includes the gender of the current subject to use the correct pronouns, we want to forget the previous gender once a new subject is seen.

What new information to store in the cell state.
"Input Gate Layer" i_t returns 0 or 1 to decide which values to update.
Tanh Layer creates a vector of new candidate values $\hat{C}_t$ to add to the cell state.



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

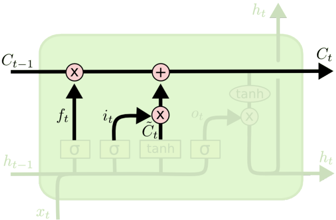
source

Example: Deciding to add the gender of the new subject to the cell state.

Architecture of an LSTM

Update Gate

Upate the old cell state C_{t-1} into the new one C_t.
Multiply the old state C_{t-1} by f_t, forgetting things decided early on.
Add scaled new candidate values decided by the input gate layer $i_t \times \hat{C}_t$.

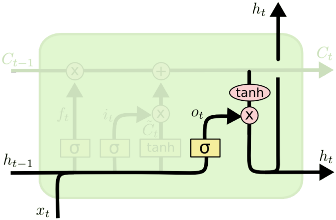


$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

source

Example: Where we actually drop the info about the old subject's gender and add the new subject's gender.

Output based on a filtered version of the current cell state.
o_t returns 0 or 1 to decide which parts of the cell state to output.
Tanh layer to push the values of the current cell state between -1 and 1, and multiply it by the output of the sigmoid layer.



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

source

Example: Since it just saw a subject, output information relevant to a verb, such as if the subject is singular or plural so we know what form the verb should be.