Tools & Models for Data Science

Deep Neural Networks (4): LSTM

Chris Jermaine & Risa Myers

Rice University

Objectives

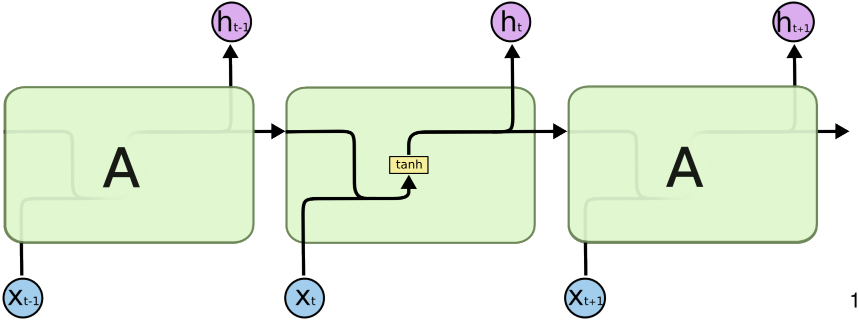
- Learn about the issues with RNNs and learn about a neural network architecture that solves some of these issues

RNN

- Have a chain of repeating “modules”
  - Each is a simple feed-forward network
  - Often a single, fully-connected layer
  - Each neuron uses some standard activation function (tanh, logistic)

Unrolling RNNs

- Training/prediction works via unrolling
- Input at each time tick concatenated with last internal state
  - Sent through a tanh/logistic layer
  - Result of that layer used to produce output, sent into next tick (Jordan network)
  - Each module is 1 iteration of the network



1

[1] Reprinted with permission. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Problem With RNNs

- The "vanishing gradient" problem
- During back-propagation
  - Update magnitude drops exponentially with distance from output
  - Recall

$$\frac{\partial L}{\partial w_{i,j,k}} = \frac{\partial L}{\partial y_{i,k}} \frac{\partial y_{i,k}}{\partial v_{i,k}} \frac{\partial v_{i,k}}{\partial w_{i,j,k}}$$

$$\frac{\text{Loss}}{\text{Neuron Output}} \; \frac{\text{Neuron Output}}{\text{Neuron input}} \; \frac{\text{Neuron Input}}{\text{Weights}}$$

  - If activation function is logistic function σ then $\frac{\partial y_{i,k}}{\partial v_{i,k}} = \sigma(v_{i,k})(1 - \sigma(v_{i,k}))$
  - Means you have a multiplier that maxes out at 0.25
  - Max value is when $v_{i,k} = 0.5$
  - $\sigma(0.5) = 0.25$
  - As you back-propagate, you keep multiplying by this in DP... $.25 \times .25 \times .25 \ldots$ gradient gets tiny
- Result is that output at time tick t...
  - ...has little interaction with input at tick $t - 100$ during back-propagation
  - Practically speaking: means back-propagation has limited-duration memory

LSTM

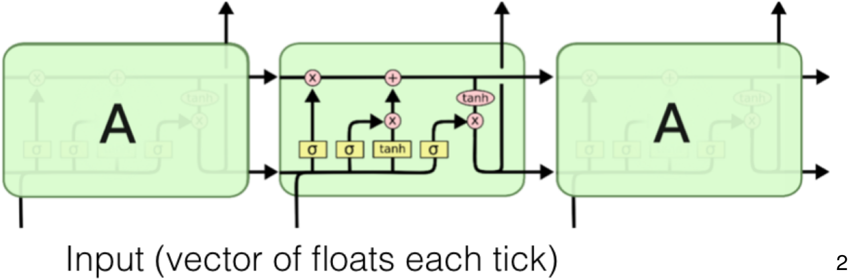
- First proposed in 1997 by Hichreiter/Schimiduber
  - Uses a more complicated architecture in each recurrent unit
  - Now used in many state-of-the-art sequence-based ML
  - Translation, text processing, speech-to-text, many others...

No Vanishing Gradients?

- Key idea: have a state that passes from tick to tick
  - Explicitly decide which part of state to update
  - Which part of the state to forget
  - And which part of the state affects the output
- If “forget” is turned off
  - State just passes through un-modified
  - Derivative of identity function is 1
  - So gradient updates during back-propagation pass through un-modified
  - No vanishing gradients!

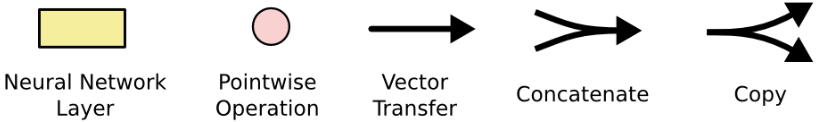
Basic LSTM Architecture

- An LSTM-based RNN is unrolled into a sequence of LSTM “modules”
  - Each module accepts input at a particular time tick
  - As well as state (“long term” memory) and last output (“short term” memory)
  - Uses those to output a view of its current state

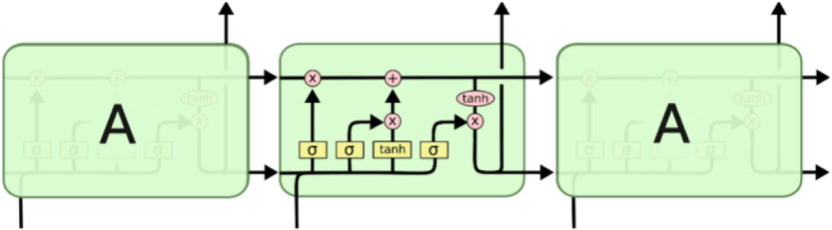


[2] Reprinted with permission. http://colah.github.io/posts/2015-08-Understanding-LSTMs/

A Note on Notation



- σ means a logistic activation layer at the top
- tanh means a hyperbolic tangent activation layer at the top

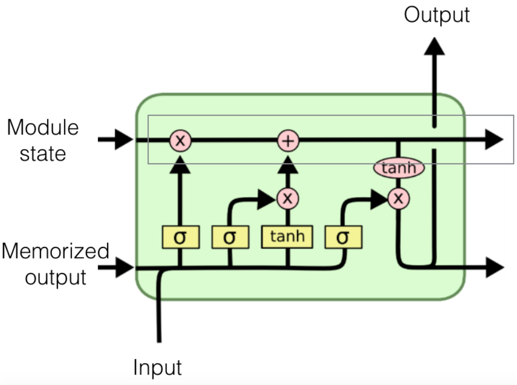


Input (vector of floats each tick)[3]

[3] Reprinted with permission. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Now, Let's Go Over the Parts in Detail

- At the top is a state (long term memory) that passes through the unit
  - Possibly unimpeded
  - Possibly modified

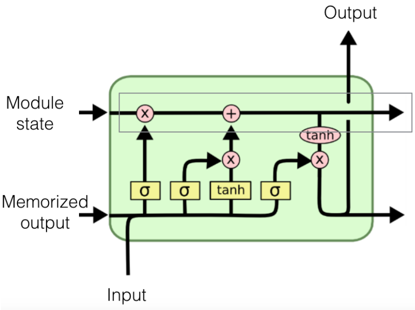


[a]

[a]Reprinted with permission.
`http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Changing Long-Term Memory Contents

- There are two operations that can affect the state...
  - An item-by-item vector multiplication
  - Known as the “forget gate”
  - And an item-by-item vector addition
  - Known as the “input gate”

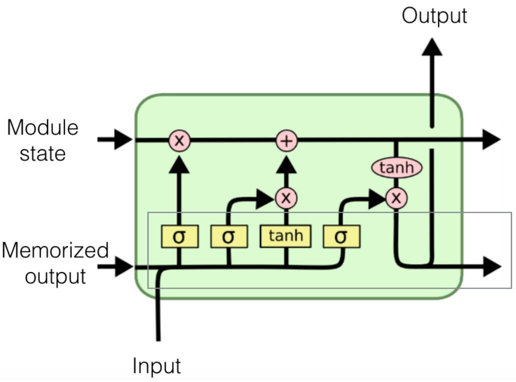


[a]

[a] Reprinted with permission.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

At the Bottom...

- Memorized output and new input flow though the unit
  - They control the forget gate and the input gate
  - Might see new input and decide not to modify state
  - Or might decide to throw out old state and rebuild
- At the end...
  - Input and state of memory used to produce output

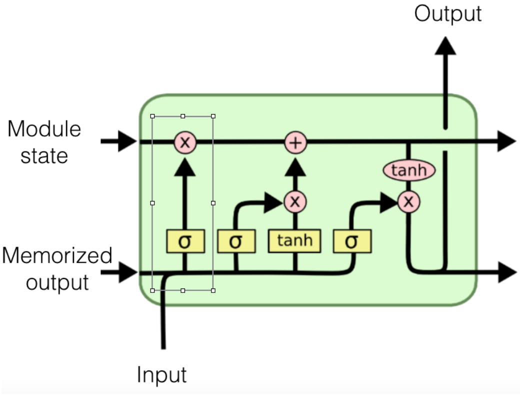


[a]

[a] Reprinted with permission.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Now Let's Walk Through Input Processing

- First we run the "forget gate"
  - Last output and new input pass through a NN
  - With a logistic layer at top
  - Produces all values from 0 to 1
  - Item-by-item multiply with state

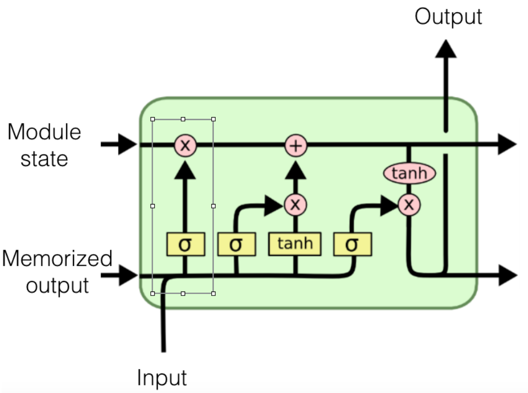


[a]

[a]Reprinted with permission.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Forget Gate

- Controls what items in state are forgotten
  - Logistic produces a 0 for a dimension?
  - Then you will zero-out that dimension in the input

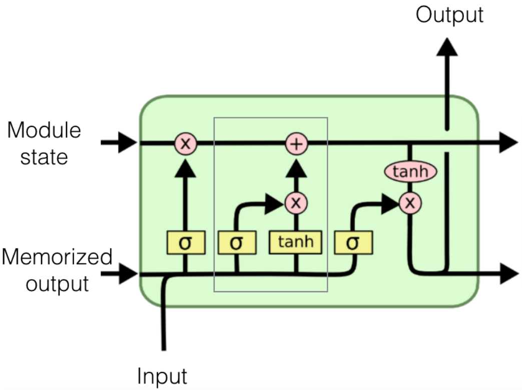


a

[a] Reprinted with permission. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Next Is the “Input Gate”

- Input plus last output passes through two NNs
  - One with a tanh layer at the top... produces activations from -1 to +1
  - One with a logistic layer at the top... produces activations from 0 to 1
- Item-by-item multiply produces a vector of updates to state
  - This update is added into the long term memory
  - This is the “input gate”

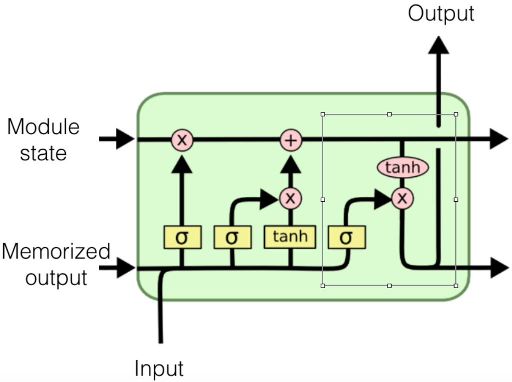


[a]

---

[a]Reprinted with permission.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- Now we have modified the state (long-term memory)
- Use the state to produce the output
  - First, push state through a tanh layer
  - Maps memory to values from -1 to +1

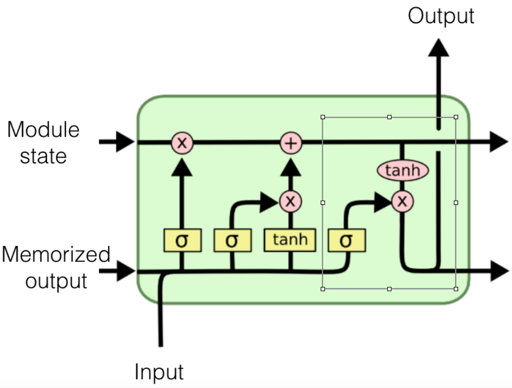


a

[a]Reprinted with permission.
`http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Output Gate

- And push the input and last output through a final NN
  - With a logistic activation at top
  - Produces values between 0 and 1
  - Item-by-item multiply with post-processed state
  - This decides what part of the state to output
  - Called the "output gate"



[a]

[a] Reprinted with permission.
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

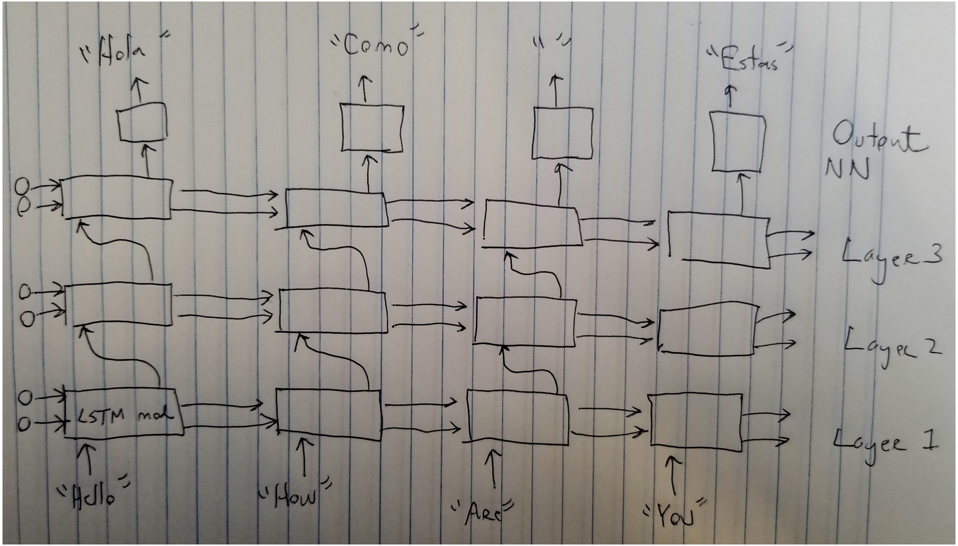
Output Typically Pushed Through a Final NN

- Output is simply a subset of the long-term memory
  - Like a brain dump
  - Simply reports encoded contents of the memory
  - Not directly useful to solve any task
  - Need to post-process (decode) it
- For example, if task is part-of-speech (POS) tagging
  - We might push output through a NN
  - Topped with a tanh layer
  - One neuron for each POS (noun, verb, adjective, etc.)
  - Push neuron outputs through a softmax
  - Output of softmax chooses final POS

Stacking These Modules

- Say we want some more learning capacity
  - Can “stack” LSTMs
- At each time tick
  - Have m stacked LSTM modules
  - Creates a stack with m layers
  - Each with its own set of parameters
  - LSTM in layer l has same internal NN params, no matter the time tick

Stacking LSTMs



Creates a Grid of LSTMs

- LSTM layer l at time tick t
  - Sends its state and output to LSTM module at layer l, time tick $t+1$
  - Sends its output to LSTM module at layer $l+1$, time tick t
- LSTM layer 1 at time tick t
  - Gets its input externally, from input stream
  - All other layers get their input from previous layer at tick t
- The output from the last/top layer at tick t...
  - Is used to produce output at tick t

Uses for Stacked LSTM Modules

- Recognize more complex concepts/objects
- Learn hierarchies
- Predict activity from multiple signals

Wrap up

? How can we use what we learned today?

- What LSTMs are and how they work

? What do we know now that we didn't know before?

- Another way to handle sequential data in a neural network