What is num_units in tensorflow BasicLSTMCell?

Ask Question

12

In the mnist lstm example here , i don't understand what the hidden layer means ? Is it the imaginary-layer formed when you represent an unrolled RNN over time ? why is the num_units = 128. I know i should read colah's blog in detail to understand this but before that i just want to get some code working for a sample time series data i have.

tensorflow lstm recurrent-neural-network

9

share edit flag

asked Jun 18 '16 at 19:51

jakeN
106 3 11

add a comment

start a bounty

2 Answers

active oldest votes

10

The number of hidden units is a direct representation of the learning capacity of a neural network -- it reflects the number of *learned parameters*. The value `128` was likely selected arbitrarily or empirically. You can change that value experimentally and rerun the program to see how it affects the training accuracy (you can get better than 90% test accuracy with *a lot* fewer hidden units). Using more units makes it more likely to perfectly memorize the complete training set (although it will take longer, and you run the risk of over-fitting).

The key thing to understand, which is somewhat subtle in the famous Colah's blog post (find "*each line carries an entire vector*"), is that **`X` is an *array* of data** (nowadays often called a *tensor*) -- it is not meant to be a *scalar* value. Where, for example, the `tanh` function is shown, it is meant to imply that the function is *broadcast* across the entire array (an implicit `for` loop) -- and not simply performed once per time-step.

As such, the *hidden units* represent tangible storage within the network, which is manifest primarily in the size of the *weights* array. And because an LSTM actually does have a bit of it's own internal storage separate from the learned model parameters, it has to know how many units there are -- which ultimately needs to agree with the size of the weights. In the simplest case, an RNN has no internal storage -- so it doesn't even need to know in advance how many "hidden units" it is being applied to.

- A good answer to a similar question here.
- You can look at the source for BasicLSTMCell in TensorFlow to see exactly how this is used.

Side note: This notation is very common in statistics and machine-learning, and other fields that process large batches of data with a common formula (3D graphics is another example). It takes a bit of getting used to for people who expect to see their `for` loops written out explicitly.

share edit flag

edited Apr 13 at 12:44

Community ♦
1 1

answered Sep 11 '16 at 20:13

nobar
17.9k 8 71 76

Further questions: How much total memory is involved? How are the weights connected to the LSTM units? Note: See TensorBoard graph visualizations. – nobar Sep 11 '16 at 21:12

I recommend LSTM: A Search Space Odyssey sections 1-3. – nobar Sep 12 '16 at 6:15

Looks like there was a followup in the comments here: RNNS IN TENSORFLOW, A PRACTICAL GUIDE AND UNDOCUMENTED FEATURES – nobar Sep 12 '16 at 13:24

Did I get it right: "a simple RNN doesn't need to know in advance how many hidden units"? Doesn't it need to know that to construct the weights that map between the units -- which grow in count exponentially based on the number of units (even in the simplest RNN). I think that I didn't understand that aspect of the architecture when I wrote this answer (see my first comment). But note that graph visualizations don't tend to help due to the array-based notation. – nobar Dec 17 '16 at 14:50

...Kind of funny that, using an array-based notation, a data path with an exponential signal count can be represented by a single dark line. – nobar Dec 17 '16 at 15:01

add a comment

7

The argument `n_hidden` of `BasicLSTMCell` is the number of hidden units of the LSTM.

As you said, you should really read Colah's blog post to understand LSTM, but here is a little heads up.

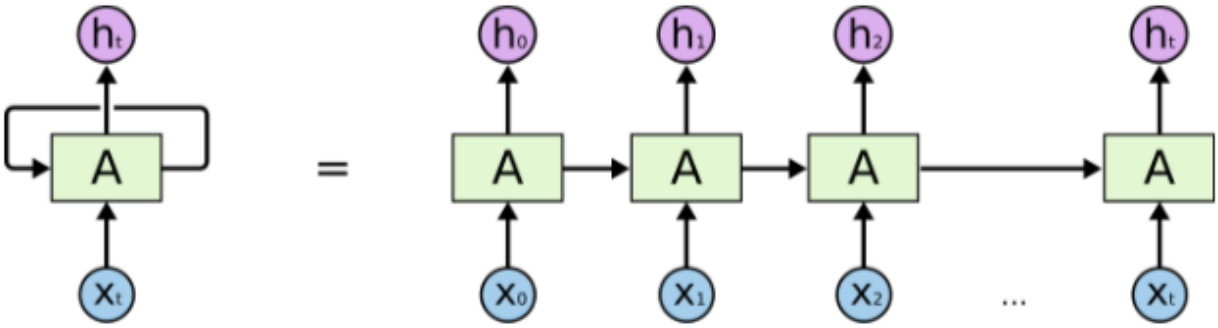
If you have an input `x` of shape `[T, 10]` , you will feed the LSTM with the sequence of values from `t=0` to `t=T-1` , each of size `10` .

At each timestep, you multiply the input with a matrix of shape `[10, n_hidden]` , and get a `n_hidden` vector.

Your LSTM gets at each timestep `t` :

- the previous hidden state `h_{t-1}` , of size `n_hidden` (at `t=0` , the previous state is `[0., 0., ...]`)
- the input, transformed to size `n_hidden`
- it will **sum** these inputs and produce the next hidden state `h_t` of size `n_hidden`

From Colah's blog post:



An unrolled recurrent neural network.

If you just want to have code working, just keep with `n_hidden = 128` and you will be fine.

share edit flag

answered Jun 18 '16 at 21:25

Olivier Moindrot
10.4k 3 33 53

1 "the input, transformed to size n_hidden" is totally cool when done like you say, with matrix multiplication. But in the mnist code example i mentioned, he is seems to be juggling all the vectors values in the batch at : x = tf.transpose(x,

[1, 0, 2]) ... , to get 28 x 128 x 28 shape. I dont get that . – jakeN Jun 19 '16 at 0:42

The RNN iterates over each row of the image. In the code of the `RNN` function, they want to get a list of length `128` (the number of steps, or number of rows of the image), with each element of shape `[batch_size, row_size]` where `row_size=28` (size of a row of the image). – Olivier Moindrot Jun 19 '16 at 9:53

Is there an upper limit to the input layer size in tf ? I get segfault when increasing the dimension to thousand plus and its fine with less. Also , shouldn't it be "...they want to get a list of length 28... " there ^ – jakeN Jun 26 '16 at 5:27

Yes you are right it should be `28` . The only limit to the size of the input is the memory of your GPU. If you want to use higher input dimension, you should adapt your batch size so that it fits into your memory – Olivier Moindrot Jun 26 '16 at 13:51

add a comment