# How do we approximately calculate how much memory is required to run a program?

Asked 2 years, 7 months ago  Modified 6 months ago  Viewed 3k times

0

1

Today I was trying to implement an object detection API in Tensorflow. After carrying out the training process, I was trying to run the program to detect objects in webcam. As I was running it, the following message was printed in the terminal:

> Allocator (GPU_0_bfc) ran out of memory trying to allocate 2.05GiB with freed_by_count=0. The caller indicates that this is not a failure, but may mean that there could be performance gains if more memory were available

Due to the performace issue it seems I am getting a lot of false positives.

How can we calculate beforehand the memory required to run this program, or any program?

I am not asking how much memory it is using, which we can find out. I am using Python.

python tensorflow memory memory-management

Share Follow

edited Jan 21, 2020 at 10:58
ForceBru
41.7k • 10 • 62 • 89

asked Dec 11, 2019 at 9:06
Fasty
754 • 10 • 33

Are you asking about predicting the memory requirements of a random program, in the general case, without any additional information about it? Or are you interested specifically in the memory requirements of running TensorFlow, given a specific network? – DeducibleSteak Jan 21, 2020 at 11:12

The latter one! :) – Fasty Jan 21, 2020 at 11:45

## 2 Answers

Sorted by: Highest score (default)

Trending sort available

3

In Object Detection, most of the Layers used will be CNNs and the Calculation of Memory Consumption for CNN is explained below. You can follow the same approach for other layers of the Model.

> For example, consider a convolutional layer with 5 × 5 filters, outputting 200 feature maps of size 150 × 100, with stride 1 and SAME padding. If the input is a 150 × 100 RGB image (three channels), then the number of parameters is (5 × 5 × 3 + 1) × 200 = 15,200 (the +1 corresponds to the bias terms), which is fairly small compared to a fully connected layer.7 However, each of the 200 feature maps contains 150 × 100 neurons, and each of these neurons needs to compute a weighted sum of its 5 × 5 × 3 = 75 inputs: that's a total of 225 million float multiplications. Not as bad as a fully con-nected layer, but still quite computationally intensive. Moreover, if the feature maps are represented using 32-bit floats, then the convolutional layer's output will occupy 200 × 150 × 100 × 32 = 96 million bits (about 11.4 MB) of RAM.8 And that's just for one instance! If a training batch contains 100 instances, then this layer will use up over 1 GB of RAM!

More understanding about Memory Consumption can be found from the below Question and the respective Answer:

**Question**:

> Consider a CNN composed of three convolutional layers, each with 3 × 3 kernels, a stride of 2, and SAME padding. The lowest layer outputs 100 feature maps, the middle one outputs 200, and the top one outputs 400. The input images are RGB images of 200 × 300 pixels. What is the total number of parameters in the CNN? If we are using 32-bit floats, at least how much RAM will this network require when making a prediction for a single instance? What about when training on a mini-batch of 50 images?

**Answer is mentioned below**:

> Let's compute how many parameters the CNN has. Since its first convolutional layer has 3 × 3 kernels, and the input has three channels (red, green, and blue), then each feature map has 3 × 3 × 3 weights, plus a bias term. That's 28 parame- ters per feature map. Since this first convolutional layer has 100 feature maps, it has a total of 2,800 parameters. The second convolutional layer has 3 × 3 kernels, and its input is the set of 100 feature maps of the previous layer, so each feature map has 3 × 3 × 100 = 900 weights, plus a bias term. Since it has 200 feature maps, this layer has 901 × 200 = 180,200 parameters. Finally, the third and last convolutional layer also has 3 × 3 kernels, and its input is the set of 200 feature maps of the previous layers, so each feature map has 3 × 3 × 200 = 1,800 weights, plus a bias term. Since it has 400 feature maps, this layer has a total of 1,801 × 400 = 720,400 parameters. All in all, the CNN has 2,800 + 180,200 + 720,400 = 903,400 parameters. Now let's compute how much RAM this neural network will require (at least) when making a prediction for a single instance. First let's compute the feature map size for each layer. Since we are using a stride of 2 and SAME padding, the horizontal and vertical size of the feature maps are divided by 2 at each layer (rounding up if necessary), so as the input channels are 200 × 300 pixels, the first layer's feature maps are 100 × 150, the second layer's feature maps are 50 × 75, and the third layer's feature maps are 25 × 38. Since 32 bits is 4 bytes and the first convolutional layer has 100 feature maps, this first layer takes up 4 x 100 × 150 × 100 = 6 million bytes (about 5.7 MB, considering that 1 MB = 1,024 KB and 1 KB = 1,024 bytes). The second layer takes up 4 × 50 × 75 × 200 = 3 million bytes (about 2.9 MB). Finally, the third layer takes up 4 × 25 × 38 × 400 = 1,520,000 bytes (about 1.4 MB). However, once a layer has been computed, the memory occupied by the previous layer can be released, so if everything is well optimized, only 6 + 9 = 15 million bytes (about 14.3 MB) of RAM will be required (when the second layer has just been computed, but the memory occupied by the first layer is not released yet). But wait, you also need to add the memory occupied by the CNN's parameters. We computed earlier that it has 903,400 parameters, each using up 4 bytes, so this adds 3,613,600 bytes (about 3.4 MB). The total RAM required is (at least) 18,613,600 bytes (about 17.8 MB).

For more information, refer "Memory Requirements" Section of "Chapter 13, Convolutional Neural Networks" of the Book, "Hands on Machine Learning with Scikit-Learn and Tensorflow" (pdfs are availble online).