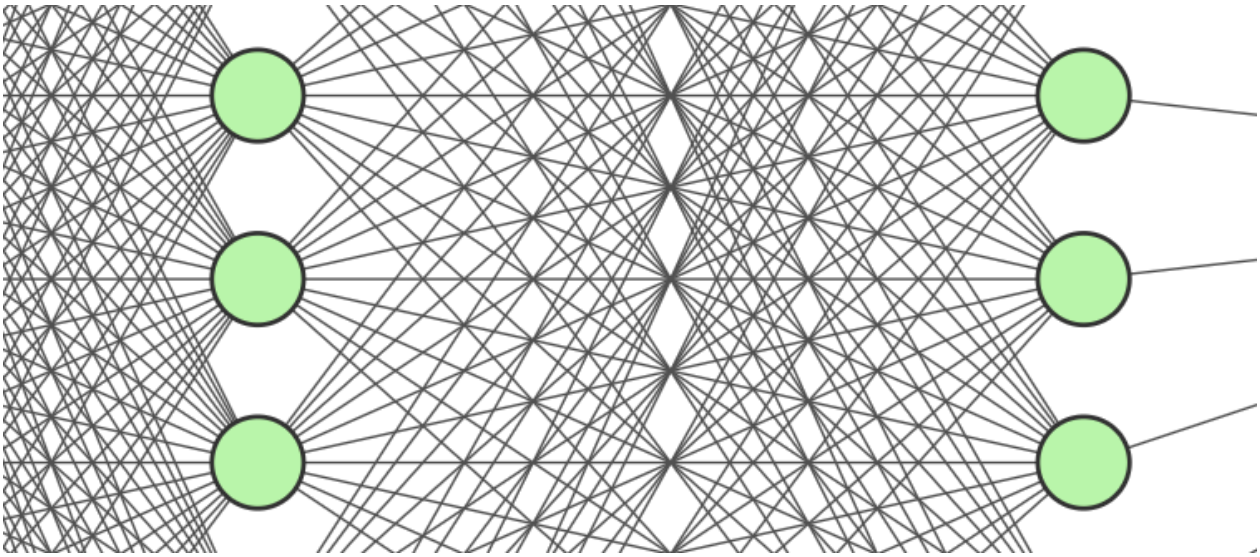
Aldo Zaimi



ML-001: How to determine the number of trainable parameters in a fully connected neural network

Introduction

In neural networks, the **number of trainable parameters** (also called **weights**) is an important hyper-parameter for several reasons. Designing a very deep/dense network (i.e. a high number of weighs) can help you learn more complex models for non-trivial tasks (e.g. advanced computer vision recognition/classification tasks), but it can also lead to severe **overfitting** if your dataset is small or your task at hand is simple. Overfitting happens when your network memorizes the dataset (instead of learning the patterns of it). Therefore, monitoring the number of trainable parameters of your networks is important during neural network design.

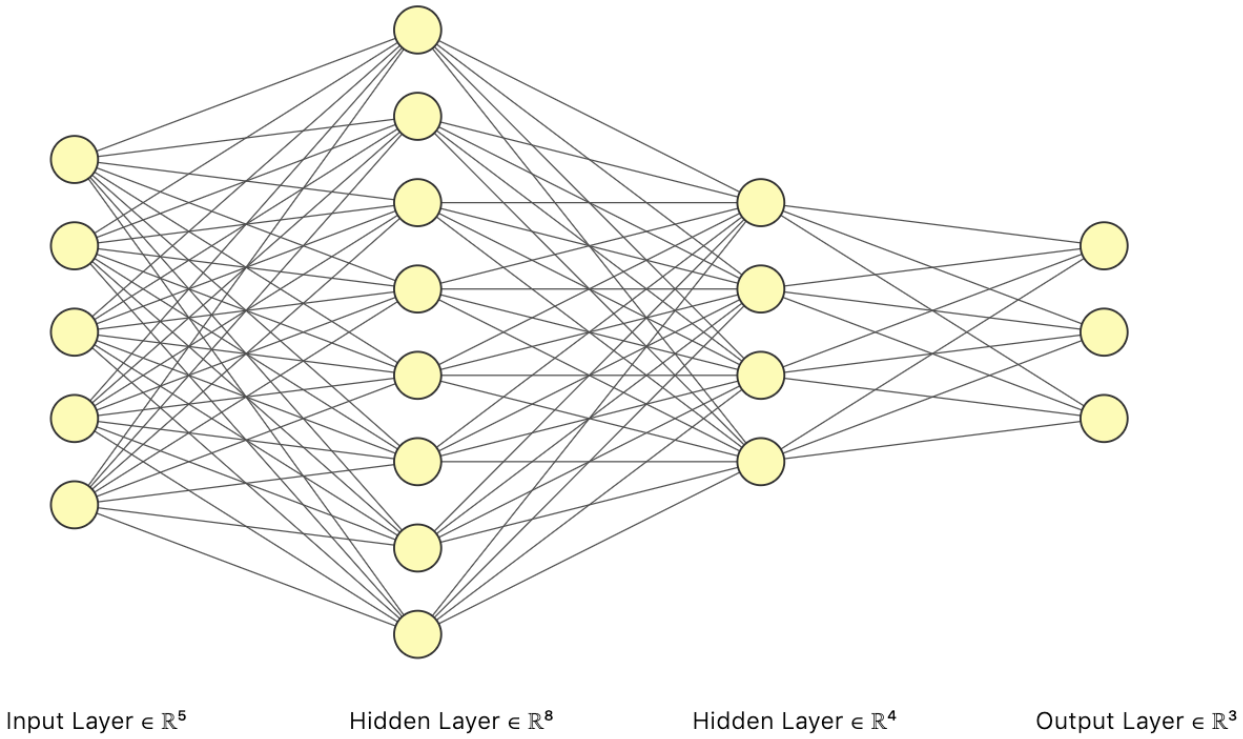
Obviously, those who have played a little with the standard deep learning frameworks know that the number of trainable parameters can easily be obtained from the code. However, it is good to have an in-

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy

Close and accept

Use Case: A Simple Fully Connected Neural Net

Let's say you want to design a standard fully connected network to classify weather data points of **5 input features** into **3 classes** (sunny, cloudy, rainy). How many trainable parameters will your architecture have if you decide to use **2 hidden layers of 8 and 4 units** respectively? Let's get a visual representation of the network to help us. (By the way, if you're looking for an easy tool to draw your neural networks, I suggest this one.)



We have **5 input units** (features), **3 output units** (because this is a 3-class classification task), a **first hidden layer with 8 units**, and a **second hidden layer with 4 units**. Intuitively, we know that in a fully connected neural net, every given unit is connected to all the units of the previous layer and to all the units of the following layer.

Let's start counting the **trainable parameters between the input layer and the first hidden layer**. Each of the 8 units of the first hidden layer is a function of the 5 input units (i.e. with 5 weights associated to the 5 connections/edges between one unit of the first hidden layer and the 5 units of the input layer). So we would have 5×8 = 40 trainable parameters between the first 2 layers. However, each of the 8 units of the first hidden layer also has a bias term. Hence, we have 40 weights and 8 bias terms between the input and the first hidden layers.

In a similar way, we can compute the number of **trainable parameters between the 2 hidden layers** as

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy

- Trainable parameters between input layer and first hidden layer: 5×8 + 8 = 48.
- Trainable parameters between first and second hidden layers: 8×4 + 4 = 36.
- Trainable parameters between second hidden layer and output layer: 4×3 + 3 = 15.
- Total number of trainable parameters of the neural net: 48 + 36 + 15 = 99.

There is a simple rule for computing the number of trainable parameters between 2 fully connected layers.

> For the fully connected layers, the number of trainable parameters can be computed by ***(n + 1)*** × ***m***, where ***n*** is the number of input units and ***m*** is the number of output units. The +1 term in the equation takes into account the bias terms.

Exercise

Determine the number of trainable parameters of the following neural net:

- Input layer: 4 units.
- Hidden layer 1: 16 units.
- Hidden layer 2: 8 units.
- Hidden layer 3: 4 units.
- Output layer: 2 units.

Solution: 262 trainable parameters.

PUBLISHED BY

Aldo Zaimi

I'm an avid AI and machine learning learner and I love sharing ideas/tutorials with people.

View all posts by Aldo Zaimi →

February 13, 2020 Aldo Zaimi deep learning, machine learning, neural networks AI, artificial intelligence, deep learning, logistic regression, machine learning, neural networks

Privacy & Cookies: This site uses cookies. By continuing to use this website, you agree to their use.
To find out more, including how to control cookies, see here: Cookie Policy