Heaton Research

The Number of Hidden Layers

2017-06-01 (/2017/06/01/hidden-layers.html) ai (/categories/ai/)

This is a repost/update of previous content that discussed how to choose the number and structure of hidden layers for a neural network. I first wrote this material during the "pre-deep learning" era of neural networks. Deep neural networks have somewhat changed the more classical recommendations of having at most 2 layers and how to choose the number of hidden layers.

A single hidden layer neural networks is capable of [universal approximation]https://en.wikipedia.org/wiki/Universal_approximation_theorem (https://en.wikipedia.org/wiki/Universal_approximation_theorem). The universal approximation theorem states that a feed-forward network, with a single hidden layer, containing a finite number of neurons, can approximate continuous functions with mild assumptions on the activation function. The first version of this theorem was proposed by Cybenko (1989) for sigmoid activation functions. Hornik (1991) expanded upon this by showing that it is not the specific choice of the activation function, but rather the multilayer feedforward architecture itself which allows neural networks the potential of being universal approximators.

Due to this theorem you will see considerable literature that suggests the use of a single hidden layer. This all changed with Hinton, Osindero, & Teh (2006). If a single hidden layer can learn any problem, why did Hinton et. al. invest so heavily in deep learning? Why do we need deep learning at all? While the universal approximation theorem states/proves that a single layer neural network *can* learn anything, it does not specify how easy it will be for that neural network to actually learn something. Every since the multilayer perceptron, we've had the ability to create deep neural networks. We just were not particularly good at training them until Hinton's groundbreaking research in 2006 and subsequent advances that built upon his seminal work.

Traditionally, neural networks only had three types of layers: hidden, input and output. These are all really the same type of layer if you just consider that input layers are fed from external data (not a previous layer) and output feed data to an external destination (not the next layer). These three layers are now commonly referred to as dense layers. This is because every neuron in this layer is fully connected to the next layer. In the case of the output layer the neurons are just holders, there are no forward connections. Modern neural networks have many additional layer types to deal with. In addition to the classic dense layers, we now also have dropout, convolutional, pooling, and recurrent layers. Dense layers are often intermixed with these other layer types.

This article deals with dense laeyrs. When considering the structure of dense layers, there are really two decisions that must be made regarding these hidden layers: how many hidden layers to actually have in the neural network and how many neurons will be in each of these layers. We will first examine how to determine the number of hidden layers to use with the neural network.

Problems that require more than two hidden layers were rare prior to deep learning. Two or fewer layers will often suffice with simple data sets. However, with complex datasets involving time-series or computer vision, additional layers can be helpful. The following table summarizes the capabilities of several common layer architectures.

Table: Determining the Number of Hidden Layers

Num Hidden Layers	Result
none	Only capable of representing linear separable functions or decisions.
1	Can approximate any function that contains a continuous mapping from one finite space to another.
2	Can represent an arbitrary decision boundary to arbitrary accuracy with rational activation functions and can approximate any smooth mapping to any accuracy.
>2	Additional layers can learn complex representations (sort of automatic feature engineering) for layer layers.

Deciding the number of hidden neuron layers is only a small part of the problem. You must also determine how many neurons will be in each of these hidden layers. This process is covered in the next section.

The Number of Neurons in the Hidden Layers

Deciding the number of neurons in the hidden layers is a very important part of deciding your overall neural network architecture. Though these layers do not directly interact with the external environment, they have a tremendous influence on the final output. Both the number of hidden layers and the number of neurons in each of these hidden layers must be carefully considered.

Using too few neurons in the hidden layers will result in something called underfitting. Underfitting occurs when there are too few neurons in the hidden layers to adequately detect the signals in a complicated data set.

Using too many neurons in the hidden layers can result in several problems. First, too many neurons in the hidden layers may result in overfitting. Overfitting occurs when the neural network has so much information processing capacity that the limited amount of information contained in the training set is not enough to train all of the neurons in the hidden layers. A second problem can occur even when the training data is sufficient. An inordinately large number of neurons in the hidden layers can increase the time it takes to train the network. The amount of training time can increase to the point that it is impossible to adequately train the neural network. Obviously, some compromise must be reached between too many and too few neurons in the hidden layers.

I have a few rules of thumb that I use to choose hidden layers. There are many rule-of-thumb methods for determining an acceptable number of neurons to use in the hidden layers, such as the following:

- The number of hidden neurons should be between the size of the input layer and the size of the output layer.
- The number of hidden neurons should be 2/3 the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

These three rules provide a starting point for you to consider. Ultimately, the selection of an architecture for your neural network will come down to trial and error. But what exactly is meant by trial and error? You do not want to start throwing random numbers of layers and neurons at your network. To do so would be very time consuming.

References

- Cybenko, G. (1989) "Approximations by superpositions of sigmoidal functions", *Mathematics of Control, Signals, and Systems*, 2 (4), 303-314
- Kurt Hornik (1991) "Approximation Capabilities of Multilayer Feedforward Networks", *Neural Networks*, 4(2), 251–257. doi:10.1016/0893-6080(91)90009-T
- Hinton, G. E.; Osindero, S.; Teh, Y. W. (2006). "A Fast Learning Algorithm for Deep Belief Nets" (PDF). Neural Computation. 18 (7): 1527–1554.

Presented at International Joint Conference on Neural Networks (IJCNN 2017) (/2017/05/25/ijcnn_2017.html)

My Review of Computer Science Ph.D. at Nova Southeastern University (/2017/07/05/finished-nsu-phd.html)

Eric Martin • a year ago

For the neuron count rules of thumb, would those be applied to each layer or a total distributed among all the layers?

1 • Reply • Share ›

Yonten Dubjur • 6 months ago

how would i cite you?

• Reply • Share ›

Jeff Heaton ➚ Yonten Dubjur • 6 months ago

This is basically a restatement of information in my deep learning book. "Artificial Intelligence for Humans, Volume 3: Deep Learning and Neural Networks" ISBN: 1505714346

• Reply • Share ›

General Useage • 2 years ago

Amazing method thank you for letting us know.
But just one question, wouldn't it be a good idea to choose the hyper parameter after analyzing the type of data that we are dealing with? Such as are there correlation among the given data etc...

1 • Reply • Share ›

ALSO ON HEATON RESEARCH

I am Starting a Ph.D. in Computer Science

1 comment • 4 years ago

papaKenya — I like your concept, after studying something new you write a book, publish or write a program. Provide more modes to purchase your books. My country has

My Experience with the Coursera Johns Hopkins Data Science …

7 comments • 4 years ago

Daniel Maxwell — That's a very good review.. Well Done !!

My First Kaggle Competition

5 comments • 4 years ago

jeffheatondotcom — No it was more of just a multi-class classification problem.

Care and Feeding of a Python Environment

1 comment • a year ago

Salomon Kabongo Kabenamualu — Thanks Professor. Very useful post

About

Jeff Heaton, Ph.D. is a computer scientist, data scientist, and indie publisher. Heaton Research is the homepage for his projects and research.

Categories

ai (/categories/ai/) (4)
aifh (/categories/aifh/) (1)
bbs (/categories/bbs/) (1)
course (/categories/course/) (1)
datascience (/categories/datascience/) (11)
encog (/categories/encog/) (1)
gpu (/categories/gpu/) (2)
java (/categories/java/) (1)
kaggle (/categories/kaggle/) (1)
learning (/categories/learning/) (2)
mergelife (/categories/mergelife/) (2)
phd (/categories/phd/) (7)
presentation (/categories/presentation/) (1)
python (/categories/python/) (1)
r (/categories/r/) (4)
tensorflow (/categories/tensorflow/) (3)
wustl (/categories/wustl/) (1)

Archives

September 2019 (/archives/2019/09/) (2)
May 2019 (/archives/2019/05/) (1)
March 2019 (/archives/2019/03/) (1)
December 2018 (/archives/2018/12/) (6)
November 2018 (/archives/2018/11/) (2)
October 2018 (/archives/2018/10/) (2)
September 2018 (/archives/2018/09/) (2)
August 2018 (/archives/2018/08/) (1)
January 2018 (/archives/2018/01/) (1)
November 2017 (/archives/2017/11/) (3)
September 2017 (/archives/2017/09/) (1)
August 2017 (/archives/2017/08/) (1)
July 2017 (/archives/2017/07/) (4)
June 2017 (/archives/2017/06/) (1)
May 2017 (/archives/2017/05/) (1)
March 2017 (/archives/2017/03/) (1)
February 2017 (/archives/2017/02/) (1)
January 2017 (/archives/2017/01/) (1)
September 2016 (/archives/2016/09/) (1)
March 2016 (/archives/2016/03/) (1)
February 2016 (/archives/2016/02/) (1)
September 2015 (/archives/2015/09/) (1)
August 2015 (/archives/2015/08/) (1)
May 2015 (/archives/2015/05/) (1)
March 2015 (/archives/2015/03/) (4)
December 2014 (/archives/2014/12/) (1)
September 2014 (/archives/2014/09/) (1)
May 2014 (/archives/2014/05/) (2)
February 2014 (/archives/2014/02/) (1)
August 2013 (/archives/2013/08/) (1)
July 2013 (/archives/2013/07/) (2)
June 2013 (/archives/2013/06/) (2)
April 2013 (/archives/2013/04/) (1)
March 2013 (/archives/2013/03/) (1)

Recents

Session 16: B/I - Multivariate Feature Engineering: Beyond Simple Data Preparation (/2019/09/09/pas-2019-features.html)
Why am I getting ImportError: No module named tensorflow? (/2019/09/03/tf-no-module-jupyter.html)
Video Release Schedule for Fall 2019 Applications of Deep Learning (/2019/05/20/2019-video-schedule.html)
My Favorite TensorFlow 2.0 Articles (as of March 2019) (/2019/03/13/tensorflow_20_articles.html)
Time Lapse: Creating Several MergeLife Cellular Automata Online with JavaScript (/2018/12/26/youtube-2018-12-26-mergelife-timelapse.html)

Jeff Heaton is a computer scientist, data scientist, and indie publisher. Heaton Research is the homepage for his projects and research.
Tips and support. (/tips.html)

GitHub (https://github.com/jeffheaton) Twitter (https://twitter.com/jeffheaton)
Youtube (https://www.youtube.com/user/HeatonResearch) Facebook (https://www.facebook.com/encog.framework/)