Cross Validated

How to choose the number of hidden layers and nodes in a feedforward neural network?

Asked 13 years, 3 months ago Modified 1 year, 2 months ago Viewed 1.1m times

804

Is there a standard and accepted method for selecting the number of layers, and the number of nodes in each layer, in a feed-forward neural network? I'm interested in automated ways of building neural networks.

model-selection neural-networks

Share Cite Improve this question

Follow

edited Mar 15, 2017 at 17:51

Post Self

113 5

asked Jul 20, 2010 at 0:15

Rob Hyndman

55.5k 28 143 194

11 Among all the great answers, i found this paper helpful dstath.users.uth.gr/papers/IJRS2009_Stathakis.pdf – Debpriya Seal Jun 24, 2017 at 1:27

1 @DebpriyaSeal not *that* useful though... – DarkCygnus Aug 16, 2017 at 18:51

3 Come to party a bit late. But this is still an open research topic as part of AutoML or AutoDL and NAS (Neural Architecture Search). There is no universal answer for this question yet. – patagonicus Jun 25, 2021 at 10:23

11 Answers

Sorted by: Highest score (default)

661

+100

I realize this question has been answered, but I don't think the extant answer really engages the question beyond pointing to a link generally related to the question's subject matter. In particular, the link describes one technique for programmatic network configuration, but that is not a "*[a] standard and accepted method*" for network configuration.

By following a small set of clear rules, one can programmatically set a competent network architecture (i.e., the number and type of neuronal layers and the number of neurons comprising each layer). Following this schema will give you a competent architecture but probably not an optimal one.

But once this network is initialized, you can iteratively tune the configuration during training using a number of ancillary algorithms; one family of these works by *pruning* nodes based on (small) values of the weight vector after a certain number of training epochs--in other words, eliminating unnecessary/redundant nodes (more on this below).

So every NN has three types of layers: *input*, *hidden*, and *output*.

Creating the NN architecture, therefore, means coming up with values for the number of layers of each type and the number of nodes in each of these layers.

The Input Layer

Simple--every NN has exactly one of them--no exceptions that I'm aware of.

With respect to the number of neurons comprising this layer, this parameter is completely and uniquely determined once you know the shape of your training data. Specifically, *the number of neurons comprising that layer is equal to the number of features (columns) in your data*. Some NN configurations add one additional node for a bias term.

The Output Layer

Like the Input layer, every NN has *exactly one* output layer. Determining its size (number of neurons) is simple; it is completely determined by the chosen model configuration.

Is your NN going to run in *Machine* Mode or *Regression* Mode (the ML convention of using a term that is also used in statistics but assigning a different meaning to it is very confusing)? Machine mode: returns a class label (e.g., "Premium Account"/"Basic Account"). Regression Mode returns a value (e.g., price).

If the NN is a regressor, then the output layer has a single node.

If the NN is a classifier, then it also has a single node unless *softmax* is used in which case the output layer has one node per class label in your model.

The Hidden Layers

So those few rules set the number of layers and size (neurons/layer) for both the input and output layers. That leaves the hidden layers.

How many hidden layers? Well, if your data is linearly separable (which you often know by the time you begin coding a NN), then you don't need any hidden layers at all. Of course, you don't need an NN to resolve your data either, but it will still do the job.

Beyond that, as you probably know, there's a mountain of commentary on the question of hidden layer configuration in NNs (see the insanely thorough and insightful NN FAQ for an excellent summary of that commentary). One issue within this subject on which there is a consensus is the performance difference from adding additional hidden layers: the situations in which performance improves with a second (or third, etc.) hidden layer are very few. *One hidden layer is sufficient for the large majority of problems.*

So what about the size of the hidden layer(s)--how many neurons? There are some empirically derived rules of thumb; of these, the most commonly relied on is '*the optimal size of the*

hidden layer is usually between the size of the input and size of the output layers'. Jeff Heaton, the author of Introduction to Neural Networks in Java, offers a few more.

In sum, for most problems, one could probably get decent performance (even without a second optimization step) by setting the hidden layer configuration using just two rules: (i) the number of hidden layers equals one; and (ii) the number of neurons in that layer is the mean of the neurons in the input and output layers.

Optimization of the Network Configuration

Pruning describes a set of techniques to trim network size (by nodes, not layers) to improve computational performance and sometimes resolution performance. The gist of these techniques is removing nodes from the network during training by identifying those nodes which, if removed from the network, would not noticeably affect network performance (i.e., resolution of the data). (Even without using a formal pruning technique, you can get a rough idea of which nodes are not important by looking at your weight matrix after training; look at weights very close to zero--it's the nodes on either end of those weights that are often removed during pruning.) Obviously, if you use a pruning algorithm during training, then begin with a network configuration that is more likely to have excess (i.e., 'prunable') nodes--in other words, when deciding on network architecture, err on the side of more neurons, if you add a pruning step.

Put another way, by applying a pruning algorithm to your network during training, you can approach optimal network configuration; whether you can do that in a single "up-front" (such as a genetic-algorithm-based algorithm), I don't know, though I do know that for now, this two-step optimization is more common.

Share Cite Improve this answer

Follow

edited Aug 31, 2022 at 12:09

Johan_A_M

75 1 9

answered Aug 2, 2010 at 2:20

doug

10.4k 1 25 26

52 You state that for the majority of problems need only one hidden layer. Perhaps it is better to say that NNs with more hidden layers are extremly hard to train (if you want to know how, check the publications of Hinton's group at Uof Toronto, "deep learning") and thus those problems that require more than a hidden layer are considered "non solvable" by neural networks. – bayerj Jul 12, 2011 at 12:50

26 You write *If the NN is a regressor, then the output layer has a single node.*. Why only a single node? Why can't I have multiple continuous outputs? – gerrit Nov 12, 2012 at 15:46

9 @gerrit You can definitely have multiple continuous outputs if your target output is vector-valued. Defining an appropriate loss function for vector-valued outputs can be a bit trickier than with one output though. – lmjohns3 Aug 29, 2013 at 23:23

7 I thought it was the opposite than this: *If the NN is a classifier, then it also has a single node unless softmax is used in which case the output layer has one node per class label in your model.* – dawid Jan 17, 2014 at 1:04

10 @doug Thank you for this wonderful answer. This allowed me to reduce my ANN from 3 hidden layers down to 1 and achieve the same classification accuracy by setting the right number of hidden

neurons... I just used the average of the input and output summed together. Thanks! – rayryeng Feb 7, 2017 at 21:58

206

@doug's answer has worked for me. There's one additional rule of thumb that helps for supervised learning problems. You can usually prevent over-fitting if you keep your number of neurons below:

$$N_h = \frac{N_s}{(\alpha * (N_i + N_o))}$$

N_i = number of input neurons.
N_o = number of output neurons.
N_s = number of samples in training data set.
α = an arbitrary scaling factor usually 2-10.

Others recommend setting α to a value between 5 and 10, but I find a value of 2 will often work without overfitting. You can think of α as the effective branching factor or number of nonzero weights for each neuron. Dropout layers will bring the "effective" branching factor way down from the actual mean branching factor for your network.

As explained by this excellent NN Design text, you want to limit the number of free parameters in your model (its degree or number of nonzero weights) to a small portion of the degrees of freedom in your data. The degrees of freedom in your data is the number samples * degrees of freedom (dimensions) in each sample or $N_s * (N_i + N_o)$ (assuming they're all independent). So α is a way to indicate how general you want your model to be, or how much you want to prevent overfitting.

For an automated procedure you'd start with an α of 2 (twice as many degrees of freedom in your training data as your model) and work your way up to 10 if the error (loss) for your training dataset is significantly smaller than for your test dataset.

Share Cite Improve this answer
Follow

edited May 5, 2021 at 14:03

Ari Cooper-Davis
135 6

answered Feb 6, 2015 at 7:22

hobs
2,235 1 14 11

12 This formula is very interesting and helpful. Is there is any reference for this formula? It would be more helpful. – prashanth Feb 16, 2016 at 14:14

2 @prashanth I combined several assertions and formulas in the NN Design text referenced above. But I don't think it's explicitly called out in the form I show. And my version is a very crude approximation with a lot of simplifying assumptions. So YMMV. – hobs Feb 16, 2016 at 17:30

1 First I wanted to write training set instead of test set in previous comment. Maybe this formula makes sense if we are to read it as "you need at least that many neurons to learn enough features (the DOF you mentioned) from dataset". If the features of dataset are representative of population and how well the model can generalize maybe it's a different question (but an important one). – kon psych Feb 22, 2016 at 22:07

5 Are you sure this is a good estimate for networks with more than one hidden layer? Isn't it the case than for multiple hidden layers the number of parameters is much greater than $N_h \cdot (N_i + N_o)$?

– Mateusz May 24, 2017 at 22:37

3 @mateus, perhaps a slightly better rule of thumb for multiple layers is the `N_h` (average number of hidden neurons per layer) solution to this `N_s = (N_i + N_o) * N_h ^ N_hidden_layers`. But I still wouldn't use this formula. It's only for very basic problems (toy problems) when you don't plan to implement any other regularization approaches. – hobs May 25, 2017 at 19:40

87

From ***Introduction to Neural Networks for Java*** *(second edition)* by **Jeff Heaton** - preview freely available at Google Books and previously at author's website:

The Number of Hidden Layers

There are really two decisions that must be made regarding the 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 two hidden layers are rarely encountered. However, neural networks with two hidden layers can represent functions with any kind of shape. There is currently no theoretical reason to use neural networks with any more than two hidden layers. In fact, for many practical problems, there is no reason to use any more than one hidden layer. Table 5.1 summarizes the capabilities of neural network architectures with various hidden layers.

Table 5.1: Determining the Number of Hidden Layers

```
| Number of Hidden Layers | Result |

 0 - 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.
```

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.
>
> There are many rule-of-thumb methods for determining the correct 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. Chapter 8, “Pruning a Neural Network” will explore various ways to determine an optimal structure for a neural network.

I also like **the following snippet** from an **answer I found at researchgate.net**, which conveys a lot in just a few words:

> Steffen B Petersen · Aalborg University
>
> [...]
>
> In order to secure the ability of the network to generalize the number of nodes has to be kept as low as possible. If you have a large excess of nodes, you network becomes a memory bank that can recall the training set to perfection, but does not perform well on samples that was not part of the training set.

Share Cite Improve this answer Follow

edited May 31, 2018 at 1:05

answered Nov 3, 2015 at 22:20

red-o-alf
995 6 5

1 Do you happen to know the source of the quote of Steffen B Petersen? – Sebastian Nielsen Oct 6, 2018 at 10:19

1 I am sorry I don't. I tried searching for it but I couldn't find it... I think the article has been removed from the web. Maybe you can contact him directly? – red-o-alf Oct 9, 2018 at 22:28

Shouldn't the size of the training set be taken into account? I have a tabular dataset with ~300,000 unique samples (car prices). The input layer has 89 nodes. Training a network with no regularization and only 89 nodes in a single hidden layer, I get the training loss to plateau after a few epochs. RMSE plateaus at ~$1,800 (single output node is price in this regression problem). – rodrigo-silveira Jul 9, 2019 at 2:48

2 I think the source of the quote by Steffen B Petersen was here: researchgate.net/post/... – TripleAntigen Sep 26, 2019 at 8:59

Though this answer has mostly copied stuff from a book whose reference is given in answer but it is somehow answering the question. – Rahul Jha Aug 16, 2020 at 15:28

50

I am working on an empirical study of this at the moment (approching a processor-century of simulations on our HPC facility!). My advice would be to use a "large" network and regularisation, if you use regularisation then the network architecture becomes less important (provided it is large enough to represent the underlying function we want to capture), but you do need to tune the regularisation parameter properly.

One of the problems with architecture selection is that it is a discrete, rather than continuous, control of the complexity of the model, and therefore can be a bit of a blunt instrument, especially when the ideal complexity is low.

However, this is all subject to the "no free lunch" theorems, while regularisation is effective in most cases, there will always be cases where architecture selection works better, and the only way to find out if that is true of the problem at hand is to try both approaches and cross-validate.

If I were to build an automated neural network builder, I would use Radford Neal's Hybrid Monte Carlo (HMC) sampling-based Bayesian approach, and use a large network and integrate over the weights rather than optimise the weights of a single network. However that is computationally expensive and a bit of a "black art", but the results Prof. Neal achieves suggests it is worth it!

Share Cite Improve this answer Follow

answered Sep 3, 2010 at 8:40

Dikran Marsupial
52.9k 8 136 198

4 "I am working on an empirical study of this at the moment" - Is there any update? – Martin Thoma Apr 24, 2017 at 12:16

3 no, 'fraid not, I'd still recommend large(ish) network and regularisation, but there is no silver bullet, some problems don't need regularisation, but some datasets need hidden layer size tuning as well as regularisation. Sadly reviewers didn't like the paper :-(– Dikran Marsupial Apr 24, 2017 at 13:07

19

> • Number of hidden nodes: There is no magic formula for selecting the optimum number of hidden neurons. However, some thumb rules are available for calculating the number of hidden neurons. A rough approximation can be obtained by the geometric pyramid rule proposed by Masters (1993). For a three layer network with n input and m output neurons, the hidden layer would have $\sqrt{n * m}$ neurons.

Ref:

1 Masters, Timothy. Practical neural network recipes in C++. Morgan Kaufmann, 1993.

[2] http://www.iitbhu.ac.in/faculty/min/rajesh-rai/NMEICT-Slope/lecture/c14/l1.html

Share Cite Improve this answer

Follow

edited Sep 7, 2017 at 7:13

Ferdi

5,141 9 46 64

answered Feb 16, 2016 at 14:17

prashanth

4,017 7 23 37

This rule seemed to work quite well with a number of different data sets and three hidden layers. One thing is for certain, using this rule, the number of neurons in the hidden layer(s) will be less than the number of input features (size n). – user32398 Jun 2, 2019 at 20:38

If I am using 15 input parameter which will finally return one output so as per your formula, hidden later neurons would be only 3-4 for this model. Is that so. – Rahul Jha Aug 16, 2020 at 15:48

@RahulJha yes as per the rule. But take the result with a pinch of salt. Its just one of the many options available. Go through other answers as well. – prashanth Aug 18, 2020 at 9:37

it is still a mystery for me. – Rahul Jha Aug 18, 2020 at 13:04

16

As far as I know there is no way to select automatically the number of layers and neurons in each layer. But there are networks that can build automatically their topology, like EANN (Evolutionary Artificial Neural Networks, which use Genetic Algorithms to evolved the topology).

There are several approaches, a more or less modern one that seemed to give good results was NEAT (Neuro Evolution of Augmented Topologies).

Share Cite Improve this answer

Follow

edited Feb 20, 2017 at 16:03

Ferdi

5,141 9 46 64

answered Jul 20, 2010 at 0:47

Vicente Cartas

269 1 3

I've listed many ways of topology learning in my masters thesis, chapter 3. The big categories are:

10

- Growing approaches
- Pruning approaches
- Genetic approaches
- Reinforcement Learning
- Convolutional Neural Fabrics

Share Cite Improve this answer Follow

answered Aug 1, 2017 at 6:39

Martin Thoma
1,657 1 18 34

9

Automated ways of building neural networks using global hyper-parameter search:

Input and output layers are fixed size.

What can vary:

- the number of layers
- number of neurons in each layer
- the type of layer

Multiple methods can be used for this discrete optimization problem, with the network out of sample error as the cost function.

- 1) Grid / random search over the parameter space, to start from a slightly better position
- 2) Plenty of methods that could be used for finding the optimal architecture. (Yes, it takes time).
- 3) Do some regularization, rinse, repeat.

Share Cite Improve this answer Follow

edited Apr 13, 2017 at 12:44

Community Bot
1

answered Dec 14, 2015 at 5:43

user91213
2,084 15 28

9

Sorry I can't post a comment yet so please bear with me. Anyway, I bumped into this discussion thread which reminded me of a paper I had seen very recently. I think it might be of interest to folks participating here:

> AdaNet: Adaptive Structural Learning of Artificial Neural Networks
>
> Corinna Cortes, Xavier Gonzalvo, Vitaly Kuznetsov, Mehryar Mohri, Scott Yang ; Proceedings of the 34th International Conference on Machine Learning, PMLR 70:874-883, 2017.
>
> Abstract We present a new framework for analyzing and learning artificial neural networks. Our approach simultaneously and adaptively learns both the structure of

the network as well as its weights. The methodology is based upon and accompanied by strong data-dependent theoretical learning guarantees, so that the final network architecture provably adapts to the complexity of any given problem.

Share Cite Improve this answer Follow

answered Oct 13, 2017 at 7:06
chainD
131 2 4

8

I'd like to suggest a less common but super effective method.

Basically, you can leverage a set of algorithms called "genetic algorithms" that try a small subset of the potential options (random number of layers and nodes per layer). It then treats this population of options as "parents" that create children by combining/ mutating one or more of the parents much like organisms evolve. The best children and some random ok children are kept in each generation and over generations, the fittest survive.

For ~100 or fewer parameters (such as the choice of the number of layers, types of layers, and the number of neurons per layer), this method is super effective. Use it by creating a number of potential network architectures for each generation and training them partially till the learning curve can be estimated (100-10k mini-batches typically depending on many parameters). After a few generations, you may want to consider the point in which the train and validation start to have significantly different error rate (overfitting) as your objective function for choosing children. It may be a good idea to use a very small subset of your data (10-20%) until you choose a final model to reach a conclusion faster. Also, use a single seed for your network initialization to properly compare the results.

10-50 generations should yield great results for a decent sized network.

Share Cite Improve this answer Follow

edited Dec 16, 2017 at 12:38
Sven Hohenstein
6,603 26 34 40

answered Dec 15, 2017 at 14:53
Dan Erez
251 2 6

1 Another very interesting way is Bayesian optimization which is also an extremely effective black-box optimization method for a relatively small number of parameters. arxiv.org/pdf/1206.2944.pdf – Dan Erez Dec 15, 2017 at 15:03

6

Number of Hidden Layers and what they can achieve:

0 - 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.

More than 2 - Additional layers can learn complex representations (sort of automatic feature engineering) for layer layers.

Share Cite Improve this answer

Follow

edited May 28, 2018 at 6:38

mkt

17.4k 11 66 162

answered May 28, 2018 at 4:02

sapy

177 2 3

16 Source(s) please. – *Reviewer* – Jim May 28, 2018 at 8:05

I would like that too. I think the >2 category referes to deep neural networks (or deep learning). – partizanos Dec 20, 2020 at 4:01

I found this here: heatonresearch.com/2017/06/01/hidden-layers.html Though it's not sourced well either. – Elenchus Oct 2, 2021 at 23:35

Highly active question. Earn 10 reputation (not counting the association bonus) in order to answer this question. The reputation requirement helps protect this question from spam and non-answer activity.