

Number of Neurons per Hidden Layer

Obviously the number of neurons in the input and output layers is determined by the type of input and output your task requires. For example, the MNIST task requires $28 \times 28 = 784$ input neurons and 10 output neurons. As for the hidden layers, a common practice is to size them to form a funnel, with fewer and fewer neurons at each layer—the rationale being that many low-level features can coalesce into far fewer high-level features. For example, a typical neural network for MNIST may have two hidden layers, the first with 300 neurons and the second with 100. However, this practice is not as common now, and you may simply use the same size for all hidden layers—for example, all hidden layers with 150 neurons: that’s just one hyperparameter to tune instead of one per layer. Just like for the number of layers, you can try increasing the number of neurons gradually until the network starts overfitting. In general you will get more bang for the buck by increasing the number of layers than the number of neurons per layer. Unfortunately, as you can see, finding the perfect amount of neurons is still somewhat of a black art.

A simpler approach is to pick a model with more layers and neurons than you actually need, then use early stopping to prevent it from overfitting (and other regularization techniques, especially *dropout*, as we will see in [Chapter 11](#)). This has been dubbed the “stretch pants” approach:¹² instead of wasting time looking for pants that perfectly match your size, just use large stretch pants that will shrink down to the right size.

Activation Functions

In most cases you can use the ReLU activation function in the hidden layers (or one of its variants, as we will see in [Chapter 11](#)). It is a bit faster to compute than other activation functions, and Gradient Descent does not get stuck as much on plateaus, thanks to the fact that it does not saturate for large input values (as opposed to the logistic function or the hyperbolic tangent function, which saturate at 1).

For the output layer, the softmax activation function is generally a good choice for classification tasks (when the classes are mutually exclusive). For regression tasks, you can simply use no activation function at all.

This concludes this introduction to artificial neural networks. In the following chapters, we will discuss techniques to train very deep nets, and distribute training across multiple servers and GPUs. Then we will explore a few other popular neural network architectures: convolutional neural networks, recurrent neural networks, and autoencoders.¹³

¹² By Vincent Vanhoucke in his [Deep Learning class](#) on Udacity.com.

¹³ A few extra ANN architectures are presented in [Appendix E](#).

Exercises

1. Draw an ANN using the original artificial neurons (like the ones in [Figure 10-3](#)) that computes $A \oplus B$ (where \oplus represents the XOR operation). Hint: $A \oplus B = (A \wedge \neg B) \vee (\neg A \wedge B)$.
2. Why is it generally preferable to use a Logistic Regression classifier rather than a classical Perceptron (i.e., a single layer of linear threshold units trained using the Perceptron training algorithm)? How can you tweak a Perceptron to make it equivalent to a Logistic Regression classifier?
3. Why was the logistic activation function a key ingredient in training the first MLPs?
4. Name three popular activation functions. Can you draw them?
5. Suppose you have an MLP composed of one input layer with 10 passthrough neurons, followed by one hidden layer with 50 artificial neurons, and finally one output layer with 3 artificial neurons. All artificial neurons use the ReLU activation function.
 - What is the shape of the input matrix \mathbf{X} ?
 - What about the shape of the hidden layer's weight vector \mathbf{W}_h , and the shape of its bias vector \mathbf{b}_h ?
 - What is the shape of the output layer's weight vector \mathbf{W}_o , and its bias vector \mathbf{b}_o ?
 - What is the shape of the network's output matrix \mathbf{Y} ?
 - Write the equation that computes the network's output matrix \mathbf{Y} as a function of \mathbf{X} , \mathbf{W}_h , \mathbf{b}_h , \mathbf{W}_o and \mathbf{b}_o .
6. How many neurons do you need in the output layer if you want to classify email into spam or ham? What activation function should you use in the output layer? If instead you want to tackle MNIST, how many neurons do you need in the output layer, using what activation function? Answer the same questions for getting your network to predict housing prices as in [Chapter 2](#).
7. What is backpropagation and how does it work? What is the difference between backpropagation and reverse-mode autodiff?
8. Can you list all the hyperparameters you can tweak in an MLP? If the MLP overfits the training data, how could you tweak these hyperparameters to try to solve the problem?
9. Train a deep MLP on the MNIST dataset and see if you can get over 98% precision. Just like in the last exercise of [Chapter 9](#), try adding all the bells and whistles