

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](#).