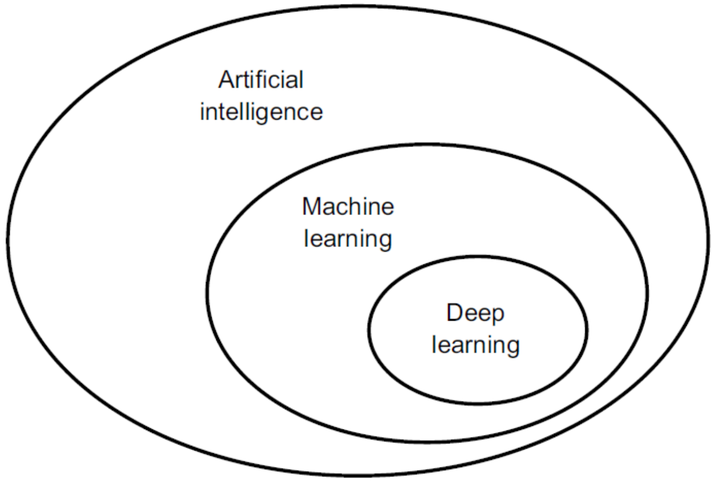
DEEP LEARNING

So what is deep learning? Deep learning is an approach to machine learning characterized by deep stacks of computations. This depth of computation is what has enabled deep learning models to disentangle the kinds of complex and hierarchical patterns found in the most challenging real-world datasets.

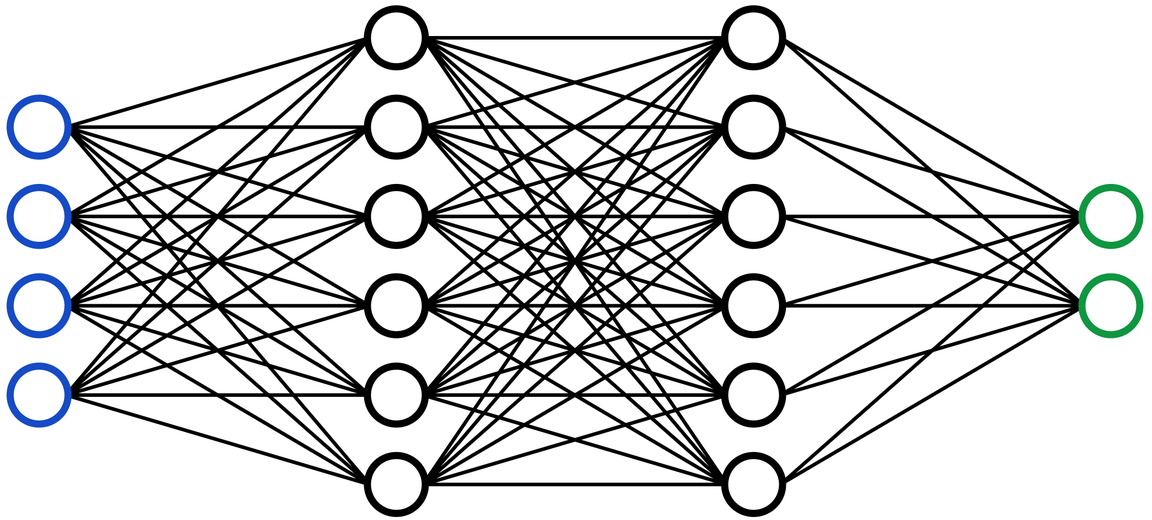


Machine Learning: The computer looks at the input data and the corresponding answers, and figures out what the rules should be.

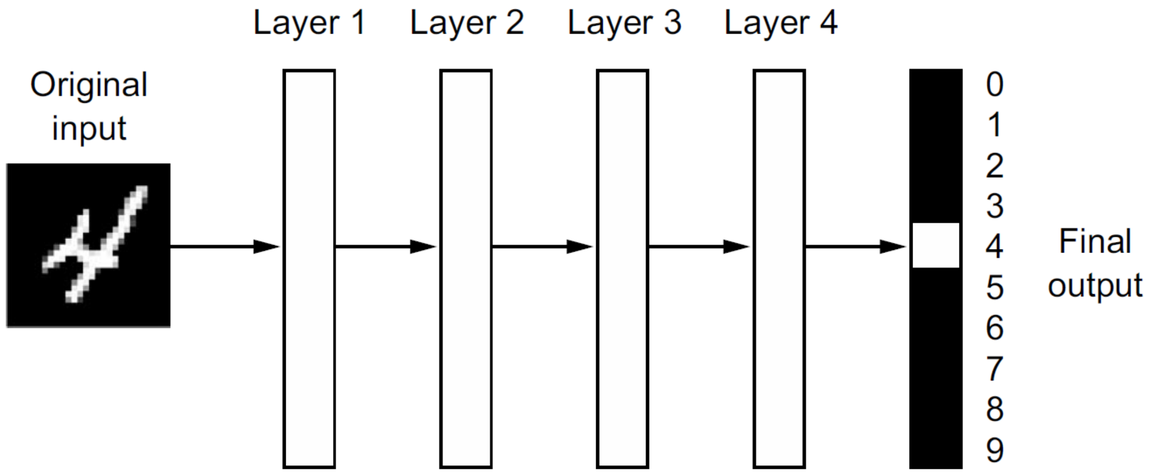
Deep learning is a specific subfield of machine learning: a new take on learning representationsfrom data that puts an emphasis on learning successive layers of increasingly meaningful representations. The “deep” in “deep learning” isn’t a reference to any kind of deeper understanding achieved by the approach; rather, it stands for this idea of successive layers of representations.

NUERAL NETWORK

Through their power and scalability neural networks have become the defining model of deep learning. Neural networks are composed of neurons, where each neuron individually performs only a simple computation. The power of a neural network comes instead from the complexity of the connections these neurons can form.



Deep neural network for digit classification



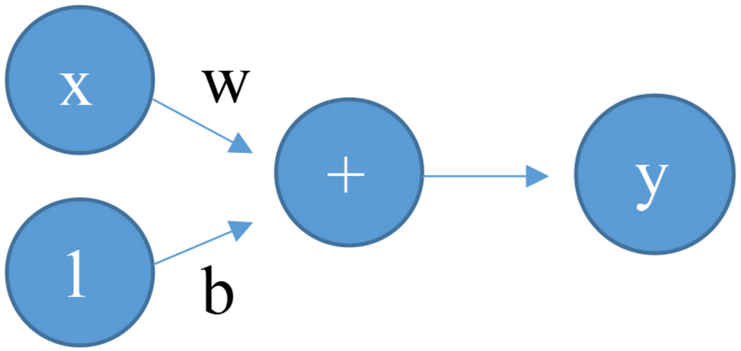
The network transforms the digit image into representations that are increasingly different from the original image and increasingly informative about the final result. You can think of a deep network as a multistage information distillation process, where information goes through successive filters and comes out increasingly purified (that is, useful with regard to some task).

NUERON

The Linear Unit

The fundamental component of a neural network: the individual neuron. As a diagram, a neuron (or unit) with one input looks like:

$$y = wx + b$$



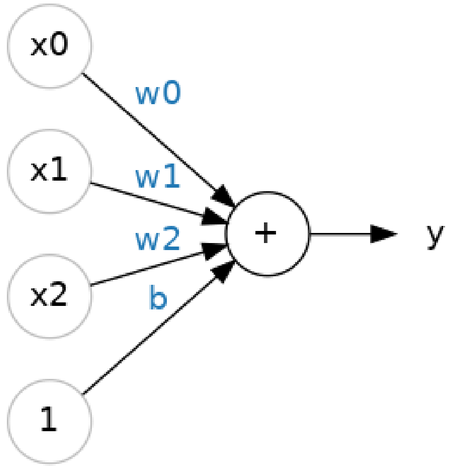
The input is x. Its connection to the neuron has a weight which is w. Whenever a value flows through a connection, you multiply the value by the connection's weight. For the input x, what reaches the neuron is w * x. A neural network "learns" by modifying its weights.

The b is a special kind of weight we call the bias. The bias doesn't have any input data associated with it; instead, we put a 1 in the diagram so that the value that reaches the neuron is just b (since 1 * b = b). The bias enables the neuron to modify the output independently of its inputs.

The y is the value the neuron ultimately outputs. To get the output, the neuron sums up all the values it receives through its connections. This neuron's activation is y = w * x + b, or as a formula y=wx+b.

MULTIPLE CONNECTIONS

We can just add more input connections to the neuron, one for each additional feature. To find the output, we would multiply each input to its connection weight and then add them all together.



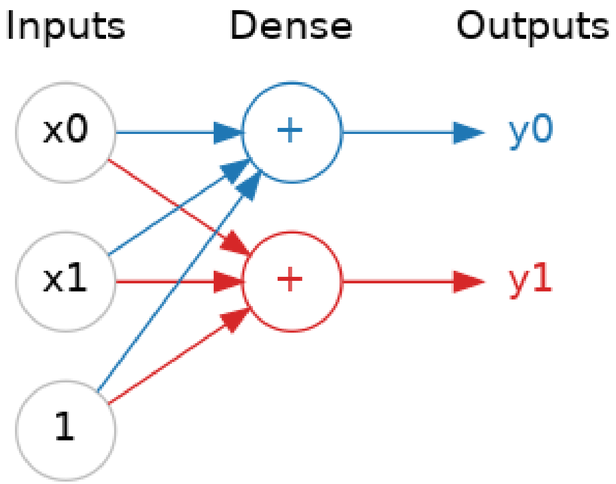
The formula for this neuron would be:

$$y=w0x0+w1x1+w2x2+b$$

A linear unit with two inputs will fit a plane, and a unit with more inputs than that will fit a hyperplane.

LAYERS OF NEURONS

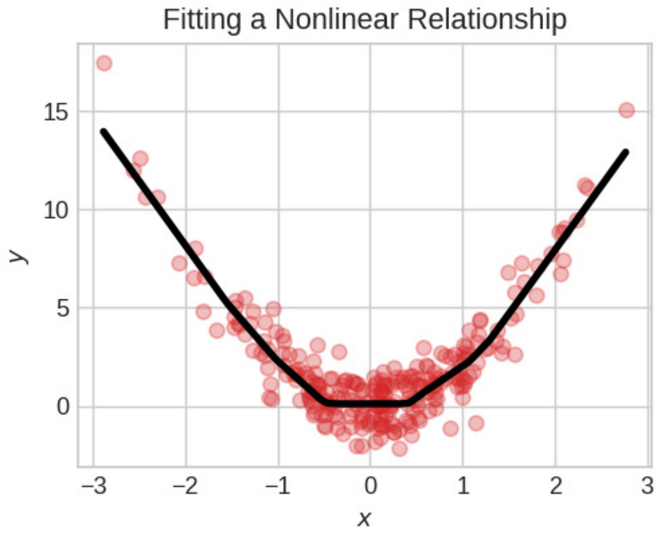
Neural networks typically organize their neurons into layers. When we collect together linear units having a common set of inputs we get a dense layer.



You could think of each layer in a neural network as performing some kind of relatively simple transformation. Through a deep stack of layers, a neural network can transform its inputs in more and more complex ways. In a well-trained neural network, each layer is a transformation getting us a little bit closer to a solution.

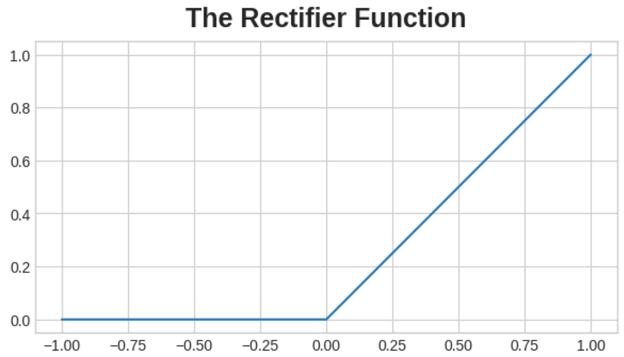
ACTIVATION FUNCTION

It turns out, however, that two dense layers with nothing in between are no better than a single dense layer by itself. Dense layers by themselves can never move us out of the world of lines and planes. What we need is something nonlinear. What we need are activation functions.



NONLINEARITY ReLu

An activation function is simply some function we apply to each of a layer's outputs (its activations). The most common is the rectifier function max(0,x).

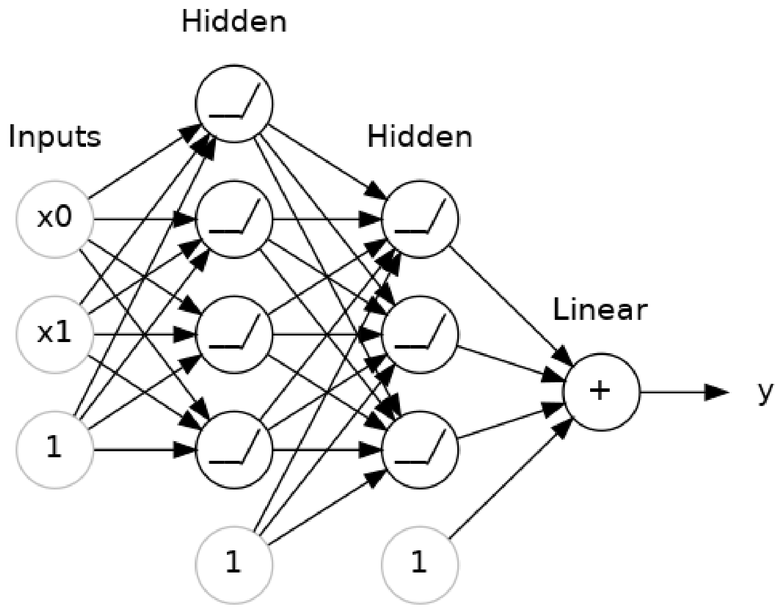


The rectifier function has a graph that's a line with the negative part "rectified" to zero. Applying the function to the outputs of a neuron will put a bend in the data, moving us away from simple lines.

When we attach the rectifier to a linear unit, we get a rectified linear unit or ReLU. (For this reason, it's common to call the rectifier function the "ReLU function".) Applying a ReLU activation to a linear unit means the output becomes max(0, w * x + b)

STACKED LAYERS

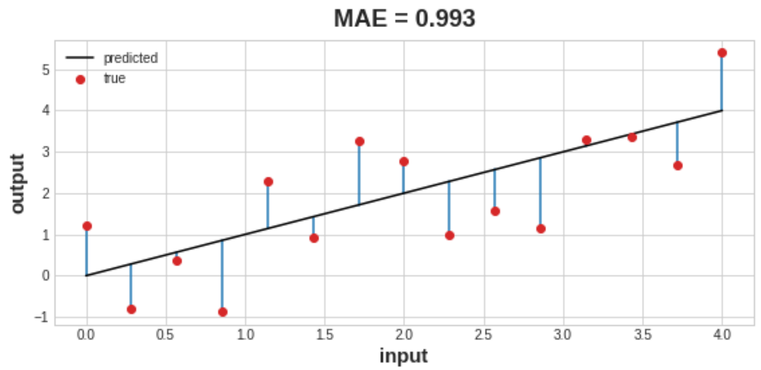
A stack of dense layers makes a “fully-connected” network.



The layers before the output layer are sometimes called hidden since we never see their outputs directly.

LOSS FUNCTION

We've seen how to design an architecture for a network, but we haven't seen how to tell a network what problem to solve. This is the job of the loss function.



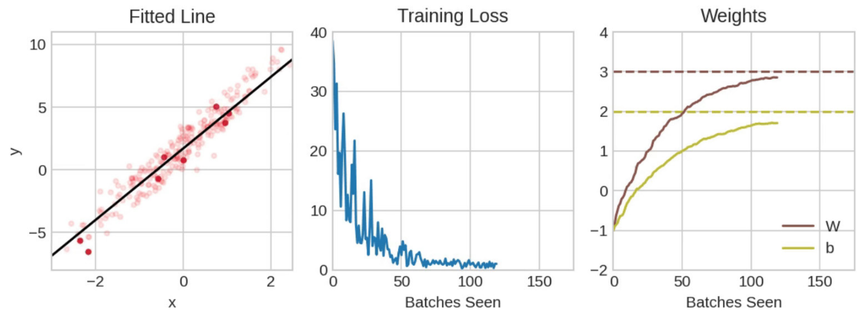
In supervised learning, there are two main types of loss functions — these correlate to the 2 major types of neural networks: regression and classification loss functions

Regression Loss Functions — used in regression neural networks; given an input value, the model predicts a corresponding output value (rather than pre-selected labels); Ex. Mean Squared Error, Mean Absolute Error

Classification Loss Functions — used in classification neural networks; given an input, the neural network produces a vector of probabilities of the input belonging to various pre-set categories — can then select the category with the highest probability of belonging; Ex. Binary Cross-Entropy, Categorical Cross-Entropy

OPTOMIZER

We've described the problem we want the network to solve, but now we need to say how to solve it. This is the job of the optimizer. The optimizer is an algorithm that adjusts the weights to minimize the loss.



Virtually all of the optimization algorithms used in deep learning belong to a family called stochastic gradient descent. They are iterative algorithms that train a network in steps. One step of training goes like this:

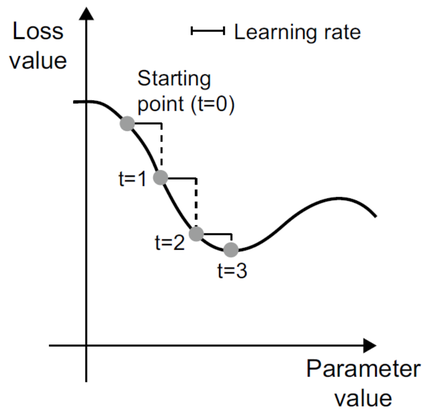
1) Sample some training data and run it through the network to make predictions.

2)Measure the loss between the predictions and the true values.

3)Finally, adjust the weights in a direction that makes the loss smaller.

Then just do this over and over until the loss is as small as you like (or until it won't decrease any further.)

LEARNING RATE



1) Draw a batch of training samples, x, and corresponding targets, y_true.

2) Run the model on x to obtain predictions, y_pred (this is called the forward pass).

3) Compute the loss of the model on the batch, a measure of the mismatch between y_pred and y_true.

4) Compute the gradient of the loss with regard to the model's parameters (this is called the backward pass).

5) Move the parameters a little in the opposite direction from the gradient—for example, W -= learning_rate * gradient—thus reducing the loss on the batcha bit. The learning rate (learning_rate here) would be a scalar factor modulating the "speed" of the gradient descent process.

BACKPROPAGATION

How do we update the weights from the ouputs of the network? We use calculus!

THE CHAIN RULE

Backpropagation is a way to use the derivatives of simple operations (such as addition, relu, or tensor product) to easily compute the gradient of arbitrarily complex combinations of these atomic operations. Crucially, a neural network consists of many tensor operations chained together, each of which has a simple, known derivative. For instance, it can be expressed as a function parameterized by the variables W1, b1, W2, and b2 (belonging to the first and second Dense layers respectively), involving the atomic operations dot, relu, softmax, and +, as well as our loss function loss, which are all easily differentiable: loss_value = loss(y_true, softmax(dot(relu(dot(inputs, W1) + b1), W2) + b2))

Calculus tells us that such a chain of functions can be derived using the following identity, called the chain rule. Consider two functions f and g, as well as the composed function fg such that fg(x) == f(g(x)):

```
def fg(x):
  x1 = g(x)
  y = f(x1)
  return y
```

FULL TRAINING LOOP