The Functions of Deep Learning

Gilbert Strang

Suppose we draw one of the digits $0, 1, \ldots, 9$. How does a human recognize which digit it is? That neuroscience question is not answered here. How can a computer recognize which digit it is? This is a machine learning question. Probably both answers begin with the same idea: *Learn from examples.*

So we start with M different images (the training set). An image is a set of p small pixels—or a vector $\boldsymbol{v} = (v_1, \ldots, v_p)$. The component v_i tells us the "grayscale" of the ith pixel in the image: how dark or light it is. So we have M images each with p features: M vectors $\boldsymbol{v}$ in p-dimensional space. For every $\boldsymbol{v}$ in that training set we know the digit it represents.

In a way, we know a function. We have M inputs in $\mathbf{R}^p$ each with an output from 0 to 9. But we don't have a "rule". We are helpless with a new input. Machine learning proposes to create a rule that succeeds on (most of) the training images. But "succeed" means much more than that: The rule should give the correct digit for a much wider set of test images, taken from the same population. This essential requirement is called *generalization*.

What form shall the rule take? Here we meet the fundamental question. Our first answer might be: $F(\boldsymbol{v})$ could be a linear function from $\mathbf{R}^p$ to $\mathbf{R}^{10}$ (a 10 by p matrix). The 10 outputs would be probabilities of each number 0 to 9. We would have $10p$ entries in the matrix and M training samples to get mostly right.

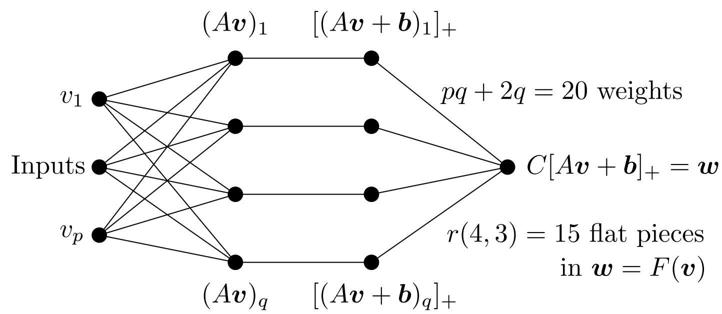
The difficulty is: Linearity is far too limited. Artistically, two zeros can make an 8. One and zero could combine into a handwritten 9 or possibly 6. Images don't add. Recognizing faces instead of numbers requires a great many pixels—and the input-output rule is nowhere near linear.

Artificial intelligence languished for a generation, waiting for new ideas. There is no claim that the absolutely best class of functions has now been found. That class needs to allow a great many parameters (called weights). And it must remain feasible to compute all those weights (in a reasonable time) from knowledge of the training set.

The choice that has succeeded beyond expectation—and has transformed shallow learning into deep learning—is *Continuous Piecewise Linear (CPL) functions.* **Linear** to preserve simplicity, **continuous** to model an unknown but reasonable rule, and **piecewise** to achieve the nonlinearity that is an absolute requirement for real images and data.

This leaves the crucial question of computability. What parameters will quickly describe a large family of CPL functions? Linear finite elements start with a triangular mesh. But specifying many individual nodes in $\mathbf{R}^p$ is expensive. It will be better if those nodes are the *intersections* of a smaller number of lines (or hyperplanes). Please know that a regular grid is too simple.

Figure 1 is a first construction of a piecewise linear function of the data vector $\boldsymbol{v}$. Choose a matrix A and vector $\boldsymbol{b}$. Then set to zero (this is the nonlinear step) all negative components of $A\boldsymbol{v}+\boldsymbol{b}$. Then multiply by a matrix C to produce the output $\boldsymbol{w} = F(\boldsymbol{v}) = C(A\boldsymbol{v} + \boldsymbol{b})_+$. That vector $(A\boldsymbol{v} + \boldsymbol{b})_+$ forms a "hidden layer" between the input $\boldsymbol{v}$ and the output $\boldsymbol{w}$.



The nonlinear function called ReLU $(x) = x_+ = \max(x, 0)$ was originally smoothed into a logistic curve like $1/(1+e^{-x})$. It was reasonable to think that continuous derivatives would help in optimizing the weights $A, \boldsymbol{b}, C$. That proved to be wrong.

The graph of each component of $(A\boldsymbol{v} + \boldsymbol{b})_+$ has two halfplanes (one is flat, from the zeros where $A\boldsymbol{v} + \boldsymbol{b}$ is negative). If A is q by p, the input space $\mathbf{R}^p$ is sliced by q hyperplanes into r pieces. We can count those pieces! This measures the "expressivity" of the overall function $F(\boldsymbol{v})$. The formula from combinatorics is:

$$r(q,p) = \binom{q}{0} + \binom{q}{1} + \cdots + \binom{q}{p}$$

This number gives an impression of the graph of F. But our function is not yet sufficiently expressive, and one more idea is needed.

Here is the indispensable ingredient in the learning function F. The best way to create complex functions from simple functions is by **composition**. Each F_i is linear (or affine) followed by the nonlinear ReLU : $F_i(\boldsymbol{v}) = (A_i\boldsymbol{v} + \boldsymbol{b}_i)_+$. Their composition is $F(\boldsymbol{v}) = F_L(F_{L-1}(\ldots F_2(F_1(\boldsymbol{v}))))$. We now have $L-1$ hidden layers before the final output layer. The network becomes deeper as L increases. That depth can grow quickly for convolutional nets (with banded Toeplitz matrices A).

The great optimization problem of deep learning is to compute weights A_i and $\boldsymbol{b}_i$ that will make the outputs $F(\boldsymbol{v})$ nearly correct—close to the digit $w(\boldsymbol{v})$ that the image $\boldsymbol{v}$ represents. This problem of minimizing some measure of $F(\boldsymbol{v}) - w(\boldsymbol{v})$ is solved by following a gradient downhill. The gradient of this complicated function is computed by *backpropagation*—the workhorse of deep learning that executes the chain rule.

A historic competition in 2012 was to identify the 1.2 million images collected in ImageNet. The breakthrough neural network in AlexNet had 60 million weights in eight layers. Its accuracy (after five days of stochastic gradient descent) cut in half the next best error rate. Deep learning had arrived.

Our goal here was to identify continuous piecewise linear functions as powerful approximators. That family is also convenient—closed under addition and maximization and composition. The magic is that the learning function $F(A_i, \boldsymbol{b}_i, \boldsymbol{v})$ gives accurate results on images $\boldsymbol{v}$ that F has never seen.

Gilbert Strang teaches linear algebra at MIT. His lectures for 18.06 and now 18.065 are on YouTube and *ocw.mit.edu*. The January 2019 textbook "Linear Algebra and Learning from Data" is described on *math.mit.edu/learningfromdata*.

References

19 Caffe : arXiv:1408.5093

5 Keras : http://keras.io/

33 MatConvNet : www.vlfeat.org/matconvnet

1 TensorFlow : www.tensorflow.org

2 Theano : arXiv : 1605.02688

6 Torch : torch.ch