Neural networks in Mathematica

Subject: Machine learning, neural networks, image processing.

Introduction

Neural networks provide a powerful (and in a sense universal) way to approximate functions. Their strength relies in two key points:

- They are essentially "model free", *i.e.* they rely on only minimal assumptions about the function that is to be approximated.
- There is a set of efficient algorithms that can be used to "learn" an approximation to the desired function directly from (a large number of) examples, *i.e.* without providing any exiplicit definition for the function itself.

Neural networks are built out of "neurons", each of which receives one or more inputs, performs an operation with them, and provides the corresponding result as its output. The strength of the connections between neurons, as well as the parameters of the operations they perform, constitute the set of parameters that can be learned to approximate the given function. They are typically initiallized to random values, and thereafter updated through (stochastic) gradient descent as the network is "trained".

Neurons are typically arranged in interconnected layers. The most basic arrengement is that of "feedforward neural networks", wherein the input layer receives the function's (properly encoded) arguments, intermediate or "hidden" layers are used to sequentially transform the input in various ways, and finally the output layer provides the (properly encoded) evaluated function.

`Mathematica` provides a flexible way to define and train neural networks; in this problem you will learn how to use it to solve a supervised classification problem.

Problem statement

Break a CAPTCHA[1] system using a neural network to learn the function that maps an image such as the one shown in Fig. 1 to the correct textual output.

Figure 1: Example challenge for the CAPTCHA system we want to break.

A CAPTCHA system is conventionally considered to be broken once the success rate of an automated solver is over 1%. For the purposes of this problem, however, we would like to achieve a success rate above 95%. Typical use of the function you define would be as follows:

```
In[1]:=SolveCAPTCHA[Import@"captcha.jpg"]
Out[1]=1 qcxvuco
```

Define a function using a neural network to break a CAPTCHA system.

Note: The `Mathematica` function `TextRecognize` provides the functionality we want, and works under-the-hood using a neural network. Give it a try and see if it meets our accuracy requirements!

Details

You are provided with two sets of images:

- The **train** dataset contains 1000 CAPTCHA's you should use to train your neural network.
- The **test** dataset contains 100 (different) CAPTCHA's you should use to test your model's accuracy.

[1] `https://en.wikipedia.org/wiki/CAPTCHA`.

Each image is provided in a JPEG file you can `Import` into `Mathematica` The filename is the correct textual output your function should produce for the corresponding image.

First, take a look at the data. Define the set of characters that are used and the region of the images where the relevant information is contained. You can discard color information converting the images to grayscale. Can you split a given image into its various characters? Even if the splitting isn't perfect (it needn't even be non-overlapping!), it will help a lot in making the neural network work properly (can you explain why?).

Now use `NetChain` to define a chain of layers you can later train to recognize single characters from a CAPTCHA image. For example, the LeNet-5 network has been successfully used to recognize digits from the MNIST database of handwritten digits[2], and will work here with minimal adaptations. It is defined in `Mathematica` by[3]:

```
In[1]:= NetChain[{
   ConvolutionLayer[20, {5, 5}],
   ElementwiseLayer[Ramp],
   PoolingLayer[{2, 2}, {2, 2}],
   ConvolutionLayer[50, {5, 5}],
   ElementwiseLayer[Ramp],
   PoolingLayer[{2, 2}, {2, 2}],
   FlattenLayer[],
   DotPlusLayer[500],
   ElementwiseLayer[Ramp],
   LinearLayer[],
   SoftmaxLayer[]
   },
  "Input" -> NetEncoder[{"Image", {28, 28},
     "Grayscale"}]
  ]
```

The LeNet–5 neural network.

[2]This database contains 28×28 grayscale images representing digits 0 through 9. You may access it within `Mathematica` using `ExampleData["MachineLearning", "MNIST"]`.

[3]This is what's shown when you run `NetModel["LeNet", "ConstructionNotebook"]`.

You can think of the neural network's output as a vector of probabilities for the input image to correspond to the various characters in the alphabet. You will then want to fix the dimensionality for the last `LinearLayer` accordingly, and to be consistent with this interpretation training data should be encoded as one-hot vectors where each position corresponds to a different character in the alphabet. You can then use a `NetDecoder` on the `"Output"` port of the chain to recover the recognized character (this will be inconsequential for training, but makes it easier to reconstruct the final answer).

Finally, train the neural network using `NetTrain`, which takes a list of training datapoints $x \mapsto f(x)$ and automatically chooses reasonable algorithms and parameters for the neural network training process. Most of these choices will be good enough, but you may want to play with the `MaxTrainingRounds` option to see how it affects accuracy. Remember to only use the **train** dataset at this stage!

Some work still needs to be done to patch everything up into a neat `SolveCAPTCHA` function. Once this is taken care of, write a function to evaluate your network against the **test** dataset, and make sure you are above the desired accuracy threshold. Compare your results to those obtained by blindly using `Mathematica` 's built-in `Classify` function.

Useful functions

You may find it useful to read the `Mathematica` help pages of the following functions:

- **Image processing functions:** `ImageDimensions`, `ImageTake`, `ColorSeparate`, ...
- **Neural network functions:** `NetChain`, `NetTrain`, `NetEncoder`, `NetDecoder`, `NetModel`, `ClassifierMeasurements`, ...
- **Other functions:** `TextRecognize`, `Characters`, `StringTake`, `Classify`, ...