Notes

**Paper: Machine Learning in Image Processing – A Survey**

Goal:

The goal of image processing is to enhance or compress image information whereas in machine learning, it is used to optimize differentiable parameters so that a certain loss or cost function is minimized.

Google Cloud Vision API – understand the content of an image

**TensorFlow – Image Recognition**

Deep convolutional neural network – can achieve reasonable performance on hard visual recognition tasks.

ImageNet – an academic benchmark for computer vision

Inception-v3 – trained for the ImageNet Large Visual Recognition Challenge. This is a standard task in computer vision, where models try to classify entire images into 1000 classes, like “Zebra:, “Dalmatian”, and “Dishwasher.”

To compare models, we examine how often the model fails to predict the correct answer as one of their top 5 guesses – termed “top-5 error rate”. AlexNet achieved by setting a top-5 error rate of 15.3% on the 2012 validation data set; Inception (GoogLeNet) achieved 6.67%; BN-Inception-v2 achieved 4.9%; Inception-v3 reaches 3.46%.

How well do humans do on ImageNet Challenge? Andrej Karpathy who attempted to measure his own performance. He reached 5.1% top-5 error rate.

Convolutional Neural Network – Look up

SoftMax Regression – Look up

**MNIST for ML Beginners**

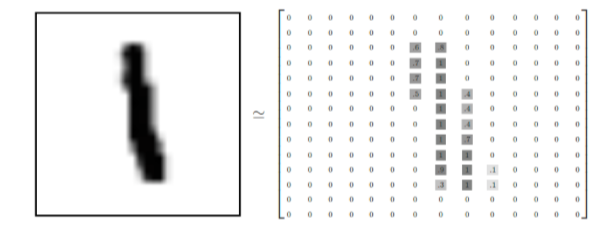
What we will accomplish in this tutorial:

* Learn about the MNIST data and softmax regressions
* Create a function that is a model for recognizing digits, based on looking at every pixel in the image
* Use TensorFlow to train the model to recognize digits by having it "look" at thousands of examples (and run our first TensorFlow session to do so)
* Check the model's accuracy with our test data

MNIST ("Modified National Institute of Standards and Technology")

Training set and Test set of digits

Each image is 28 x 28 pixels. We can interpret this as a big array of numbers:



We can flatten this array into a vector of 28 x 28 = 784 numbers. It doesn’t matter how we flatten the array as long as were consistent between images. From this perspective, the MNIST images are just a bunch of points in a 784 dimensional vector space, with a very rich structure.

Flattening the data throws away information about the 2D structure of the image. Isn’t that bad? Well, the best computer vision methods do exploit this structure and we will in later tutorials. But the simple method we will be using here, a softmax regression won’t.

The result is a tensor (an n-dimensional array) with a shape of [55000, 784]. The first dimension is an index into the list of images and the second dimension is the index for each pixel in each image. Each entry in the tensor is a pixel intensity between 0 and 1, for a particular pixel in a particular image.

Each image in MNIST has a corresponding label, a number between 0 and 9 representing the digit drawn in the image.

For the purposes of this tutorial, were going to want our labels as “one-hot vectors”. A one-hot vector is a vector which is 0 in most dimensions, and 1 in a single dimension. In this case, the nth digit will be represented as a vector which is 1 in the nth dimension. For example, 3 would be [0,0,0,1,0,0,0,0,0,0]. Consequently, mnist.train.labels is a [55000, 10] array of floats.

**Softmax Regressions**

Every image in MNIST is a handwritten digit between zero and nine. There are ten possible things an image can be. We want to look at an image and give the probabilities for it being each digit. Softmax regression is a natural simple model. If you want to assign probabilities to an object being one of several different things, softmax is the thing to do. Softmax gives us a list of values between 0 and 1 that add up to 1.

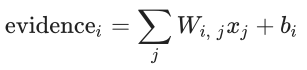
Softmax has two steps:

1. Add up the evidence of our input being in certain classes
2. Convert that evidence into probabilities.

To tally up the evidence that a given image is in a particular class, we do a weighted sum of the pixel intensities. The weight is negative if that pixel having a high intensity is evidence against the image being in that class, and positive if it is evidence in favor.

Bias

We also add some extra evidence called a bias. Basically, we want to be able to say that some things are more likely independent of the input. The result is that the evidence for a class I given an input x is:



where Wi is the weights and bi is the bias for class I and j is an index for summing over the pixels in our input image x. We then convert the evidence tallies into out predicted probabilities y using the “softmax” function:

y = softmax(evidence)

Here softmax is serving as an “activation” or “link” function, shaping the output of our linear function into the form we want – in this case, a probability distribution over 10 cases. You can think of it as converting tallies of evidence into proababilities of our input being in each class. It’s defined as:

Softmax(x) = normalize(exp(x))

More helpful to think of softmax the first way: exponentiating its inputs and then normalizing them. The exponentiation means that one more unit of evidence increases the weight given to any hypothesis multiplicatively. And conversely having one less unit of evidence means that a hypothesis gets a fraction of its earlier weight. No hypothesis ever has zero or negative weight. Softmax then normalizes these weights, so that they add up to one, forming a valid probability distribution.

**NumPy**

NumPy is the fundamental package for scientific computing with Python. It contains among other things:

* a powerful N-dimensional array object
* sophisticated (broadcasting) functions
* tools for integrating C/C++ and Fortran code
* useful linear algebra, Fourier transform, and random number capabilities

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data. Arbitrary data-types can be defined. This allows NumPy to seamlessly and speedily integrate with a wide variety of databases.

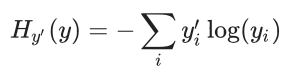
NumPy is licensed under the [BSD license](http://www.numpy.org/license.html#license), enabling reuse with few restrictions.

In order to train the model, we need to define what it means for the **model to be good**?

Well, actually in machine learning we typically define what it means for a model to be bad. We call this the cost or the loss, and it represents how far off our model is from our desired outcome. We try to minimize that error and the smaller the error margin, the better our model is.

**To determine the loss of a model: Cross-entropy function**

Cross-entropy function arises from thinking about information compressing codes in information theory but it winds up being an important idea in lots of areas, form gamblind to machine learning. Defined as:



Where y is our predicted probability distribution and y’ is the true distribution. In some rough sense, the cross entropy is measuring how efficient our predictions are for describing the truth.