Roger Ballard and Tan Qiuhao Chen

Artificial Intelligence

10/13/2016

Project Theory

**Convolution Neural Networks**

Convolutional neural networks are a special type of neural network. In order to understand what a convolutional neural network is, one must first understand the general idea behind neural networks themselves.

A Brief Introduction to Neural Networks:

Neural networks are a supervised learning approach to classification problems. Given an ample supply of training data, (and neural networks require a lot) a neural network can learn to approximate any arbitrary function. Before I get more into detail about that, and explanation of the topology of a neural network is in order.



Neural Network Topology

Image Credit: [https://commons.wikimedia.org/wiki/File:Colored\_neural\_network.svg](https://commons.wikimedia.org/wiki/File%3AColored_neural_network.svg)

A traditional neural network consists of multiple layers of perceptrons, with each layer being fully connected to the next layer. One layer is designated as the input layer, one layer is designated as the output layer, and any other layers in the middle are called hidden layers. There is a proof that a neural network with a single hidden layer with sufficient size can approximate any function to an arbitrary degree of accuracy. This is a demonstration of the power of neural networks.

![\includegraphics[width=.7\textwidth]{pics/perceptron}]()

Components of a Perceptron

Image Credit: <https://www.hiit.fi/u/ahonkela/dippa/node41.html>

A single perceptron takes some number of inputs, and produces an output. It does this by computing a weighted sum of the inputs, weighted by the learned weights of the perceptron, plus a learned bias factor. The resultant sum is passed through an activation function. This is traditionally the sigmoid function. (Other functions can be used. In order for the network to be able to learn any function, this activation function has to have certain properties. Other commonly used functions are the hyperbolic tangent function, the rectified linear function, a leaky rectified linear function, and the softplus function.)

To calculate the result for a given input, a neural network starts at the input layer, passes the values as inputs to the next layer, calculates the result of that layer, and repeats for all layers until it reaches the output layer, at which point it outputs the result.

What makes a trained neural network unique to the problem it is solving is the learned weights and biases for each of its perceptrons. These weights and biases are learned during the training stage of the neural network. In this stage, labelled data is fed to the network. (Labelled data consists of the input, as well as the desired output from the neural network.) Traditionally, neural networks are trained with a method called gradient descent. Let’s run through gradient descent as it applies to a single input. (This is called stochastic gradient descent. Another method is called batch gradient descent, which works with multiple labelled inputs at the same time, to smooth out the errors in the calculated local error gradient.)

Backpropagation starts with a forward pass through the network, to get the computed output. Then it is compared to the desired output, and a predetermined cost function is used to determine how well/poorly the two outputs match to eachother. Remember how the neural network calculated results by going through layers of functions? Well, those layers of functions can be combined together into one big function, so you can imagine a neural network as a single large elementary function, with both the inputs and the learned weights and biases as inputs. With a good choice of activation function, this function is differentiable with respect to each of the learned weights and biases. We can calculate the partial derivative of the cost function with respect to each of the learned parameters, and use this knowledge to update the parameters to lower the cost function. Repeated application of this process updates the parameters to move the cost function to a local minimum. Backpropagation doesn’t work directly with the very large elementary function; it uses a method that passes back through the network, calculating gradients as it goes along, but the concept is the same. There are many more nuances to neural network training besides what is covered in this overview; for the interested reader I recommend Chapter 10 of The Nature of Code (<http://natureofcode.com/book/chapter-10-neural-networks/>) and Chapter 2 of Neural Networks and Deep Learning (<http://neuralnetworksanddeeplearning.com/chap2.html>). These are both great resources for learning about neural networks in general, as well as many other interesting topics such as cellular automata and genetic algorithms.



Gradient Descent Visualization

Image Credit: <http://charlesfranzen.com/posts/multiple-regression-in-python-gradient-descent/>

Now, to cover what makes convolutional neural networks different from traditional neural networks. Convolutional neural networks were built to solve image processing tasks, and were inspired by research into the workings of mammal visual cortexes. In image processing, there are a large number of perceptrons in each layer, far too many to train each weight individually. Additionally, if identifying a feature in one region of a neural network is useful, then it is also likely to be useful to be able to identify that same feature in another region of an image. This leads to the solution that defines neural networks: only one sets of weights and a bias is trained for each convolutional layer. The inputs are defined by the relative position of the inputs to the location of the output. This function is applied over the whole range of the previous layer, resulting in the output of the current layer. Mathematically, this process is called convolution, which is where the technique gets its name. These special convolutional layers, along with pooling layers that reduce the layer sizes, form the core components of a convolutional neural network. The last few layers of a convolutional neural network are typically fully connected, to perform classification after the useful features have been extracted from the image and distilled.



Convolutional Layer Visualization

Image Credit: [https://commons.wikimedia.org/wiki/File:Conv\_layer.png](https://commons.wikimedia.org/wiki/File%3AConv_layer.png)

**Support Vector Machines**

Support vector machines are a type of supervised binary linear classifier. With some additional techniques, support vector machines can be modified to perform nonlinear classification, and to perform classification on an arbitrary number of classes. The idea behind support vector machines is to draw a hyperplane between two linearly separable groups of vectors. Specifically, the hyperplane which maximizes the minimum distance from the hyperplane to any point in either of the clusters. Testing points are classified by which side of the computed hyperplane they lie on. This straightforward method only works for two linearly separable classes, but techniques exists to extend support vector machines to work in more cases.



Illustration of a Basic Support Vector Machine

Image Credit: [https://commons.wikimedia.org/wiki/File:Svm\_max\_sep\_hyperplane\_with\_margin.png](https://commons.wikimedia.org/wiki/File%3ASvm_max_sep_hyperplane_with_margin.png)

Working with Non-Linearly Separable Classes:

If the two classes are not linearly separable, (no hyperplane can be drawn which perfectly divides the classes on either side) the above straightforward method does not work. In this case, a soft margin has to be created, which almost splits the data into two classes. In order to accomplish this, a cost function is computed. The main component of this cost function is the hinge loss function. The value of this function is zero if a point lies on the correct side of the hyperplane, otherwise it is proportional to how far on the wrong side of the hyperplane the point lies. The cost function is the average hinge loss function for all data points, plus a parameter that determines the tradeoff between putting more points on the correct side of the hyperplane and maximizing the minimum distance of the correct points from the hyperplane. By minimizing this cost function, a good classification hyperplane will be created. This functions closely to the basic support vector machine algorithm in the cases where the input is linearly separable, and still works well in cases where the input is not separable.



Example of a Soft-Margin Support Vector Machine

Image Credit: <http://efavdb.com/svm-classification/>

Classification with More Than Two Classes:

There are two main methods of classifying with multiple classes using support vector machines; both work by creating several binary classification support vector machines and taking a consensus vote among them to perform classification. One method works by training one support vector machine for each class. This SVM has the chosen class as one class, and every data point from every other class as the other. In other words, each sub-SVM classifies a point as either belonging to the class or belonging to some other class. The classification is chosen based on which classifier most strongly believes that the point belongs to its class. The other method creates a classifier for every pair of classes. The classification of a point is decided by which class gets voted for by the greatest number of pairwise classifiers.

Performing Non-Linear Classification:

Some data sets do not naturally fall into two groups that can be neatly separated by a hyperplane. In this case, neither the straightforward SVM algorithm nor the soft margin version will be able to perform a good classification. In this case, we can manipulate the data to exist in a higher-dimensional feature space, where the classes are neatly separable. This is called the kernel trick. For example, you can map two-dimensional points to fit on a radial Gaussian function, which puts points near the middle of the data higher than points on the outside of the data. This method is shown in the below image, and a more in-depth discussion of the technique is available at the source.



Demonstration of the Kernel Trick

Image Credit: <http://www.bindichen.co.uk/post/AI/Nonlinear-Support-Vector-Machines.html>

**Dataset**

For this project, we will be using the MNIST handwritten digit dataset. It is an image classification dataset, which is the area convolution neural networks were specifically created for. Originally, we wanted to use ImageNet, a large and complicated image recognition dataset, as our dataset. In this dataset, convolution neural networks provide state-of-the-art classification accuracies, roughly on par with human capabilities. However, this area is not particularly well-suited to support-vector machines. For this reason, we switched to MNIST, because it was simple enough that SVMs have been applied to it with reasonable success. MNIST consists of 70,000 28x28 greyscale handwritten digit images from a mix of people, split into 60,000 training images and 10,000 test images.