Abstract

In this report we train and test a set of classifiers like logistic regression and neural net with backpropagation for object recognition.After exploring these classifiers we look at deep learning techniques like Sparse Autoencoders which is a unsupervised method to learn good representation of features from images.We use the features extracted from sparse autoencoder to build a deep network for Object Recognition.

1. Introduction

An object recognition system finds object in the real world from an images of the world,using object classes which are known a priori.In this report we will discuss the different types of classification algorithms that can be used for building a model for object recognition and analyze their classification accuracy on the test set.The task is to predict the labels of images from the given categories of digits which ranges from 0-9.We first train the dataset with differnt classifiers like Neural Networks with backpropagation and Logistic Regression on the raw image pixels.We than explore unsupervised methods like Sparse Autoencoders which can learn good representation of features from the image pixels.We finally build a Deep Network using approaches we learned above for Object Recognition.

2. The Dataset

The data for the task is taken from the MNIST (Modified National Institute of Standards and Technology) dataset.The MNIST database contains 42,000 digits ranging from 0 to 9 for training the digit recognition system, and another 28,000 digits as test data. Each digit is normalized and centered in a gray-level image with size 28 28, or with 784 pixel in total as the features.

3. Training

We describe here all the training approaches we have taken to train the classifier for multiclass object detection.We also give the classification accuracy of our training algorithms on the test test.

3.1. Logistic Regression

We will be using multiple one-vs-all logistic regression model to build a multi-class classifier. Since there are 5 classes, we will need to train 5 separate logistic regression classifiers. The cost function for logistic regression is:

$$J = \frac{1}{m}\sum_{i=1}^{m}[-y^{(i)}\log(h\theta(x^{(i)})) - (1-y^{(i)})\log(1-h\theta(x^{(i)}))]$$

Here,

$$h\theta(x) = \frac{1}{1+e^{-x}}$$

We let $x_0 = 1$, so that $x \in \Re^{n+1}$ and $\theta \in \Re^{n+1}$, and θ_0 is the intercept term. We have a training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ of m examples, and the batch gradient ascent update rule is $\theta := \theta + \alpha\nabla_\theta\ell(\theta)$, where $\ell(\theta)$ is the log likelihood and $\nabla_\theta\ell(\theta)$ We thus compute the gradient:

$$\nabla_\theta\ell(\theta) = \sum_{i=1}^{m}\left(y^{(i)} - h_\theta(x^{(i)})\right)x_j^{(i)}.$$

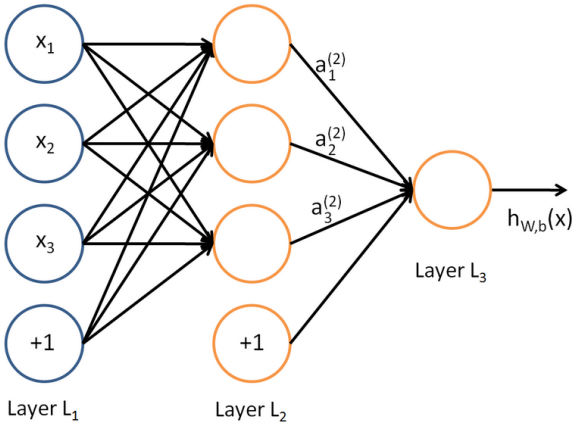
We have a training set of 500 examples.There are 240 feature vectors for each training set.Each feature represents a keypoint extracted from the image using standard algorithms.With these feature we train our clasifier using one vs all approach of Logistic Regression.After training we obtain a set of parameters which we can be used to predict the labels of the test set and can determine its accuracy.

3.2. Neural Nets with BackPropagation

We use now Neural Networks which give a way of defining a complex, non-linear form of hypotheses $h_{W,b}(x)$,with parameters W, b that we can fit to our data.To describe neural networks, we will begin by describing the simplest possible neural network, one which comprises a single "neuron." The "neuron" is a computational unit that takes as input x_1, x_2, x_3 (and a +1 intercept term), and outputs $h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$, where $f : \Re \mapsto \Re$ is called the 'activation function'.

$$f(z) = \frac{1}{1 + \exp(-z)}$$

A neural network is put together by hooking together many of our simple "neurons," so that the output of a neuron can be the input of another. For example, here is a small neural network: In this figure, we have used circles to also denote the



inputs to the network. The circles labeled "+1" are called "'bias units"', and correspond to the intercept term. The leftmost layer of the network is called the "'input layer"', and the rightmost layer the "'output layer"'.The middle layer of nodes is called the "'hidden layer"', because its values are not observed in the training set. We also say that this example neural network has 3 "'input units"' (not counting the bias unit), 3 "'hidden units"', and 1 "'output unit"'. We will let n_l denote the number of layers in our network. The neural network has parameters $(W, b) = (W^{(1)}, b^{(1)}, W^{(2)}, b^{(2)})$, where $W_{ij}^{(l)}$ denote the parameter (or weight) associated with the connection between unit j in layer l, and unit i in layer $l + 1$.Also,$b_i^{(l)}$ is the bias associated with unit i in layer $l + 1$ Given a fixed setting of the parameters W, b,our neural network defines a hypothesis $h_{W,b}(x)$ that outputs a real number.

$$\begin{aligned}
a_1^{(2)} &= f(W_{11}^{(1)} x_1 + W_{12}^{(1)} x_2 + W_{13}^{(1)} x_3 + b_1^{(1)}) \\
a_2^{(2)} &= f(W_{21}^{(1)} x_1 + W_{22}^{(1)} x_2 + W_{23}^{(1)} x_3 + b_2^{(1)}) \\
a_3^{(2)} &= f(W_{31}^{(1)} x_1 + W_{32}^{(1)} x_2 + W_{33}^{(1)} x_3 + b_3^{(1)}) \\
h_{W,b}(x) = a_1^{(3)} &= f(W_{11}^{(2)} a_1^{(2)} + W_{12}^{(2)} a_2^{(2)} + W_{13}^{(2)} a_3^{(2)} + b_1^{(2)})
\end{aligned}$$

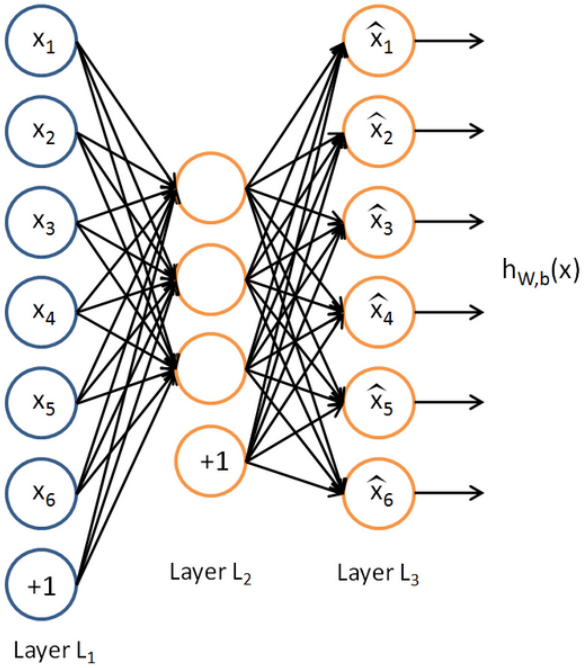
We call this step forward propagation.Neural networks can also have multiple output units. Suppose we have a fixed training set $\{(x^{(1)}, y^{(1)}), \ldots, (x^{(m)}, y^{(m)})\}$ of m training examples. We can train our neural network using batch gradient descent. In detail, for a single training example (x, y), we define the cost function with respect to that single example to be:

$$J(W, b; x, y) = \frac{1}{2} \left\| h_{W,b}(x) - y \right\|^2$$

This is a (one-half) squared-error cost function.The goal is to minimize $J(W, b)$ as a function of W and b.We use backpropagation algorithm to minimize this cost function and get optimal set of weights to make predictions.We use the same training set to train the classifier to make predictions and record its classification accuracy on the test set.

3.3. Sparse Autoencoders

Suppose we have only a set of unlabeled training examples $\{x^{(1)}, x^{(2)}, x^{(3)}, \ldots\}$, where $x^{(i)} \in \Re^n$.An autoencoder neural network is an unsupervised learning algorithm that applies backpropagation,setting the target values to be equal to the inputs. The autoencoder tries to learn a function $h_{W,b}(x) \approx x$.In other words, it is trying to learn an approximation to the identity



function, so as to output $\hat{x}$ that is similar to x.The identity function seems a particularly trivial function to be trying to learn but by placing constraints on the network, such as by limiting the number of hidden units, we can discover interesting structure about the data. Informally, we will think of a neuron as being active if its output value is close to 1, or as being inactive if its output value is close to 0.We would like to constrain the neurons to be inactive most of the time. $a_j^{(2)}$ denotes the activation of hidden unit j in the autoencoder.However,this notation doesn't make explicit what was the input x that led to that activation.Thus, we will write $a_j^{(2)}(x)$ to denote the activation of this hidden unit when the network is given a specific input x. Further, let

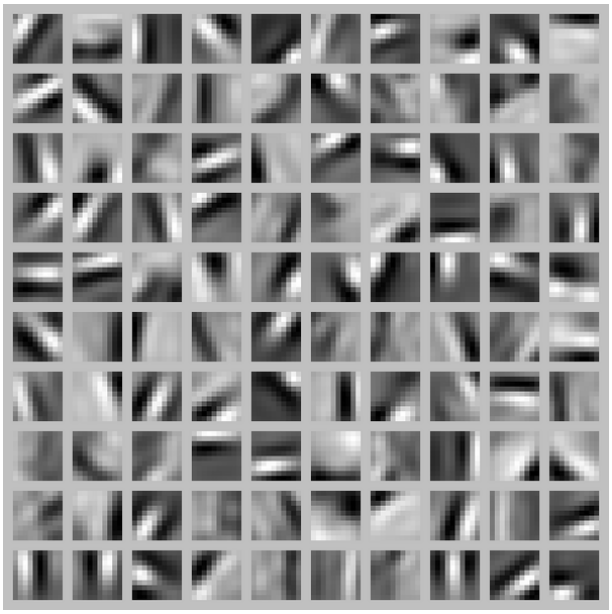
$$\hat{\rho}_j = \frac{1}{m} \sum_{i=1}^{m} \left[a_j^{(2)}(x^{(i)}) \right]$$

be the average activation of hidden unit j. We would like to (approximately) enforce the constraint $\hat{\rho}_j = \rho$, where ρ is a sparsity parameter, typically a small value close to zero .In other words, we would like the average activation of each hidden neuron j to be close to 0.05 (say).To satisfy this constraint, the hidden unit's activations must mostly be near 0. To achieve this, we will add an extra penalty term to our optimization objective that penalizes $\hat{\rho}_j$ deviating significantly from ρ.We will choose the following cost function to train sparse autoencoder and backpropagation algorithm to obtain optimal weights.

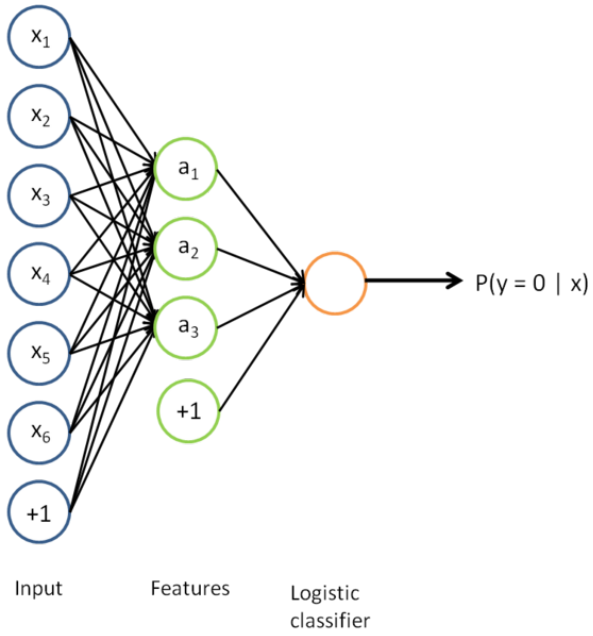
$$\sum_{j=1}^{s_2} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}.$$

3.4. Deep Networks for Classification

Using all the techniques we learnt above we now build a deep network for object classification.We will use standard MNSIT handwritten digits for our task.First we will take 10,000 examples from the data set whose label is unknown to us.We will feed this data to the sparse autoencoder.The sparse autoencoder try to learn weights to represent the image in a more compressed form.If we visualize these learned weights it looks like detecting edges at different positions in the image.



Each square in the figure above shows the input image that maximally actives one of the hidden units. We see that the different hidden units have learned to detect edges at different positions and orientations in the image.We save this learned weights to get good features from images whose labels are already known to us. We have a labeled training set that we can replace the original features with features computed by the sparse autoencoder.We do this by multiplying the weights we get from sparse autoencoder with the training set.This gives us a new training set.Finally, we train a logistic classifier to map from the features $a^{(i)}$ we laerned above to the classification label $y^{(i)}$.The overall architecture looks like below:



Here we constructed a 3-layer neural network comprising an input, hidden and output layer. While fairly effective for MNIST,this 3-layer model is a fairly shallow network which means that the features are computed using only one layer of computation. We build a deep neural networks in which we have multiple hidden layers.This will allow us to compute much more complex features of the input because each hidden layer computes a non-linear transformation of the previous layer,so a deep network can have significantly greater representational power than a shallow one.One method that has been used is the

greedy layer-wise training method.The idea is to train the layers of the network one at a time, so that we first train a network with 1 hidden layer, and only after that is done, train a network with 2 hidden layers, and so on. At each step, we take the old network with $k-1$ hidden layers, and add an additional k-th hidden layer.So here we use a 3 layer deep network.The weights that we obtain from running sparse autoencoder are again fed into a new sparse autoencoder which can learn a more compressed represntation of the learned weights.The weights from training the layers individually are then used to initialize the weights in the final deep network and only then is the entire architecture is trained together to optimize the labeled training set error.

We observe the overall performance improves significantly using deep networks.

3.5. Pre-Processing of Data

Data preprocessing plays a very important in many deep learning algorithms. In practice,we have seen our performance improved between 2-7 perecent after our data has been pre-processed before using it to train a classifier.We used simple rescaling,in which our goal was to rescale the data along each data dimension (possibly independently) so that the final data vectors lie in the range $[0, 1]$ or $[-1, 1]$.We also tried feature standardization which refers to setting each dimension of the data to have zero-mean and unit-variance. This is the most common method for normalization. We achieved this by first computing the mean of each dimension (across the dataset) and subtracts this from each dimension. Next,each dimension is divided by its standard deviation.Out of the two approaches we found z-score to perform better on RGB images.

3.6. PCA/ZCA

PCA(Principal Component Analysis) will allow us to approximate the input with a much lower dimensional one, while incurring very little error. Whitening is a preprocessing step which removes redundancy in the input, by causing adjacent pixels to become less correlated. Since grey-scale images have the stationarity property, we usually first remove the mean-component from each data example separately.After this step, PCA/ZCA whitening is often employed with a value of epsilon set large enough to low-pass filter the data.

3.7. Results

Training Method	Classification Accuracy
Logistic Regression	78%
Neural Net	88%
Deep Learning	92%

The code[1] for the all of the implementation can be found in the online repository.

References

[1] Rajat Raina,Alexis Battle,Honglak Lee,Benjamin Packer,Andrew Y. Ng *Self-taught Learning: Transfer Learning from Unlabeled Data*. Computer Science Department, Stanford University, CA 94305 USA

[2] Rajat Raina,Anand Madhavan,Andrew Y. Ng *Large-scale Deep Unsupervised Learning using Graphics Processors* . Computer Science Department, Stanford University, CA 94305 USA

[3] Yoshua Bengio *Learning Deep Architectures for AI* . Dept. IRO, Universite de Montreal C.P. 6128, Montreal, Qc, H3C 3J7, Canada

[4] Memisevic,Hinton G. E. *Unsupervised learning of image transformations.*. Computer Vision and Pattern Recognition (CVPR-07)

[1] https://github.com/manishrocksag/ObjectRecognition