LaTeX Author Guidelines for CVPR Proceedings

First Author
Institution1
Institution1 address
firstauthor@i1.org

Second Author
Institution2
First line of institution2 address
secondauthor@i2.org

Abstract

The main objective is to look at different classifiers which can be used for classifying images and analyze their performance.We then turn to Deep Learning techniques which are good at finding patterns.We build a deep network for classifying images and analyze its performance.

1. Introduction

The task is to predict the labels of images from the given categories buildings,cars, flowers,faces,shoes. We here train the dataset with differnt classifiers like Neural Networks with backpropagation,Logistic Regression, Support Vector Machines.We further take our approach to apply DeepLearning methods and compare the results with our previous approaches.We will mainly look at:

1. Sparse Autoencoders
2. SelfTaught Learning
3. Deep Networks
4. Stacked Autoencoders

2. The Dataset

The dataset consists of natural images taken from camera.The classes for the images are Buildings, Cars, Faces, Flowers and Shoes.Each image is 240 * 240 RGB image.We have 500 images for the training set and 1000 images for the test set.We have used SIFT algorithm to extract 240 feature from each of these images.For our deep learning tasks we will use samples from this raw images which are randomly picked.

3. Training

We describe here all the training approaches we have taken to train the classifier for multiclass object detection.We give brief description of all the training algorithms we have used.Finally we give the classification accuracy of this training algorithms on the test test we are using.

3.1. Logistic Regression

We will be using multiple one-vs-all logistic regression models 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:

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

Here,

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

We 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 SIFT algorithm.With these feature we train our clasifier using one vs all approach of code[1] Logistic Regression.After training we have a set of parameters which we use to predict the labels of the test set and determine its accuracy.

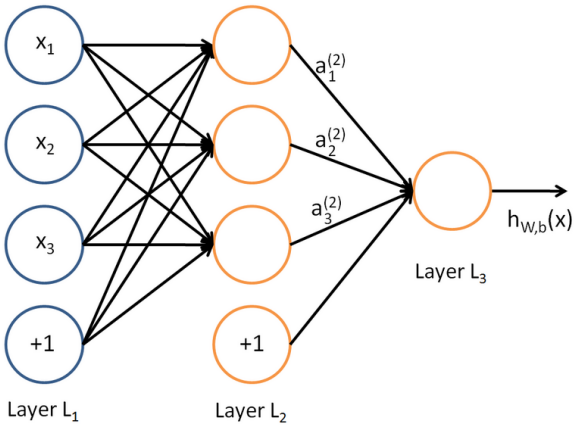
3.2. Neural Nets with BackPropagation

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 de-

[1] www.github.com



note 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.

$$a_1^{(2)} = f(W_{11}^{(1)}x_1 + W_{12}^{(1)}x_2 + W_{13}^{(1)}x_3 + b_1^{(1)})$$

$$a_2^{(2)} = f(W_{21}^{(1)}x_1 + W_{22}^{(1)}x_2 + W_{23}^{(1)}x_3 + b_2^{(1)})$$

$$a_3^{(2)} = f(W_{31}^{(1)}x_1 + W_{32}^{(1)}x_2 + W_{33}^{(1)}x_3 + b_3^{(1)})$$

$$h_{W,b}(x) = a_1^{(3)} = f(W_{11}^{(2)}a_1^{(2)} + W_{12}^{(2)}a_2^{(2)} + W_{13}^{(2)}a_3^{(2)} + b_1^{(2)})$$

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.

3.3. Support Vector Machines

A support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks.Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier. Whereas the original problem may be stated in a finite dimensional space, it often happens that the sets to discriminate are not linearly separable in that space. For this reason, it was proposed that the original finite-dimensional space be mapped into a much higher-dimensional space, presumably making the separation easier in that space. To keep the computational load reasonable, the mappings used by SVM schemes are designed to ensure that dot products may be computed easily in terms of the variables in the original space, by defining them in terms of a kernel function $K(x, y)$ selected to suit the problem.The hyperplanes in the higher-dimensional space are defined as the set of points whose dot product with a vector in that space is constant. The vectors defining the hyperplanes can be chosen to be linear combinations with parameters α_i of images of feature vectors that occur in the data base. With this choice of a hyperplane, the points x in the feature space that are mapped into the hyperplane are defined by the relation: $\sum_i \alpha_i K(x_i, x) = \text{constant}$. Note that if K(x,y) becomes small as y grows further away from x, each term in the sum measures the degree of closeness of the test point x to the corresponding data base point x_i. In this way, the sum of kernels above can be used to measure the relative nearness of each test point to the data points originating in one or the other of the sets to be discriminated. Note the fact that the set of points x mapped into any hyperplane can be quite convoluted as a result, allowing much more complex discrimination between sets which are not convex at all in the original space. Let us assume that we have n labeled examples $(x_1, y_1), \ldots, (x_n, y_n)$ with labels $y_i \in \{1, -1\}$. We want to find the hyperplane $< w, x > +b = 0$ (i.e. with parameters (w, b)) satisfying the followin three conditions:

1. The scale of (w, b) is fixed so that the plane is in canonical position w.r.t. $\{x_1, \ldots, x_n\}$. i.e.,

$$\min_{i \leq n} | < w, x_i > +b| = 1$$

2. The plane with parameters (w, b) separates the $+1$'s from the the -1's. i.e.,

$$y_i(< w, x_i > +b) \geq 0 \text{ for all } i \leq n$$

3. The plane has maximum margin $\rho = 1/|w|$. i.e., minimum $|w|^2$.

Of course there may not be a separating plane for the observed data. Let us assume, for the time being, that the data is in fact linearly separable and we'll take care of the general (more realistic) case later.

Clearly 1 and 2 combine into just one condition:

$$y_i(< w, x_i > +b) \geq 1 \text{ for all } i \leq n.$$

Thus, we want to solve the following optimization problem,

$$\text{minimize } \frac{1}{2}|w|^2$$

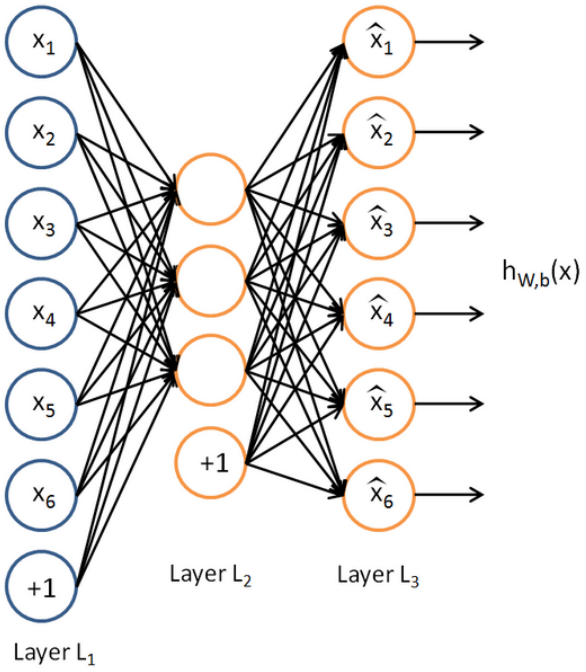
over all $w \in R^d$ and $b \in R$ subject to,

$$y_i(< w, x_i > +b) - 1 \geq 0 \text{ for all } i \leq n.$$

This is a very simple quadratic programming problem. There are readily available algorithms of complexity $O(n^3)$ that can be used for solving this problem..

3.4. 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. I.e., it uses $y^{(i)} = x^{(i)}$ Here is an autoencoder: 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" (or as "firing") 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:

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

3.5. Deep Networks for Classification

3.6. Pre-Processing of Data

3.7. References

4. Final copy

You must include your signed IEEE copyright release form when you submit your finished paper. We MUST have this form before your paper can be published in the proceedings.

5. Final copy

You must include your signed IEEE copyright release form when you submit your finished paper. We MUST have this form before your paper can be published in the proceedings.

Please direct any questions to the production editor in charge of these proceedings at the IEEE Computer Society Press: Phone (714) 821-8380, or Fax (714) 761-1784.