Journal of Control, Robotics, and Mechatronic Systems, Vol. 1 (1), 27-32, Mar. 2015
ISSN: Pending, DOI: Pending Published online: www.unitedscholars.net/jcrms-archive

Deep Learning Neural Networks for Vision and Classification in Torch7

RK Yadav, AviralTakkar

Department of Computer Engineering,Delhi Technological University, New Delhi, India.
(aviraltakkar@gmail.com)

ABSTRACT

We develop deep learning neural networks in Torch7 for two applications. First, to implement the Kaggle Forest Cover Type Prediction Challenge, which is essentially a classification problem. Based on 55 cartographic parameters of 15000 training examples, we perform supervised training of a multi-layer neural network to predict the forest cover type. We find that for learning such a large number of parameters, a multi-layer perceptron is not adequate as it is very prone to overfitting, and is not able to generalize well to data, as the number of parameters to learn increases. Thus, the need for Convolutional Neural Nets, to capture hidden relationships and dependencies among the input vector, is clearly seen. Second, based on the paper "*Convolutional Neural Networks Applied to House Numbers Digit Classification*" by LeCun et al, we implement the Convolutional Neural Network described there in Torch7 on the SVHN dataset. We achieve high accuracy in the test results.

Keywords: Convolutional Neural Networks, SVHN, Classification, Vision, Deep learning.

INTRODUCTION

We train a multi-layer perceptron for the purpose of classification challenge posed in [1]. In this challenge, you are asked to predict the forest cover type (the predominant kind of tree cover) from strictly cartographic variables (as opposed to remotely sensed data). The actual forest cover type for a given 30 x 30 meter cell was determined from US Forest Service (USFS) Region 2 Resource Information System data. Independent variables were then derived from data obtained from the US Geological Survey and USFS. The data is in raw form (not scaled) and contains binary columns of data for qualitative independent variables such as wilderness areas and soil type. There are about 15000 samples provided, each composed of 54 parameters. The total number of outcome classes are 7, hence this is a multi-class classification problem. We have implemented these problems on the Torch7 platform [4].

Through this implementation of a multi-layer perceptron, we find that such a network is inadequate to properly learn to predict a model with a large number of parameters. We found that for a high training accuracy (about 81%), we achieved very poor performance on the test data (about 36%). This anomaly is clearly due to overfitting. In statistics and machine learning, overfitting occurs when a statistical model describes random error or noise instead of the underlying relationship. Overfittinggenerally occurs when a model is excessively complex, such as having too many parameters relative to the number of observations. In our case, despite employing regularization techniques, feature scaling and using a few number parameters, the results were poor. In other words, the 'learning capacity' of the model is low. In image classification problems, where the number of parameters in the model become excessively large compared to this simple problem, we cannot use a multi-layer perceptron. Thus, we introduce Convolutional Neural Networks, and use them to tackle the second application.

For our second application, which makes use of Convolutional Neural Networks, we go a step further than character recognition. Character recognition in documents can be considered a solved task for computer vision, whether handwritten or typed. It is however a harder problem in the context of complex natural scenes like photographs where the best current methods lag behind human performance, mainly due to non-contrasting backgrounds, low resolution, de-focused and motion-blurred images and large illumination differences. [2] (Figure 1)

Figure 1. 32 by 32 cropped samples from the SVHN dataset.

We use a Convolutional Neural Network architecture for the purpose of predicting the house numbers augmented with different pooling methods and with multi stage features. In the results obtained, we achieve high training as well as testing accuracy (over 80%), thus demonstrating the high potential of ConvNets to capture hidden dependencies in images, and subsequently large predicting power.

CLASSIFICATION – KAGGLE FOREST COVER TYPE PREDICTION CHALLENGE

In this section, we discuss the model we have implemented as well as the data used. The study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. Each observation is a 30m x 30m patch. We predict an integer classification for the forest cover type. We obtained the data from [1].

The seven types are:

1. *Spruce/Fir*
2. *Lodgepole Pine*
3. *Ponderosa Pine*
4. *Cottonwood/Willow*
5. *Aspen*
6. *Douglas-fir*
7. *Krummholz*

The training set (15120 observations) contains both features and the Cover Type [1].

Data Fields

Elevation - Elevation in meters.
Aspect - Aspect in degrees azimuth.
Slope - Slope in degrees.
Horizontal_Distance_To_Hydrology - Horizontal distance to nearest surface water features.
Vertical_Distance_To_Hydrology - Vertical distance to nearest surface water features.
Horizontal_Distance_To_Roadways - Horizontal distance to nearest roadway.
Hillshade_9am (0 to 255 index) – Hill shade index at 9am, summer solstice.
Hillshade_Noon (0 to 255 index) – Hill shade index at noon, summer solstice.
Hillshade_3pm (0 to 255 index) – Hill shade index at 3pm, summer solstice.Horizontal_Distance_To_Fire_Points - Horizontal distance to nearest wildfire ignition points.
Wilderness_Area (4 binary columns, 0 = absence or 1 = presence) - Wilderness area designation.
Soil_Type (40 binary columns) - Soil Type designation.
Cover_Type (7 types, integers 1 to 7) - Forest Cover Type designation.

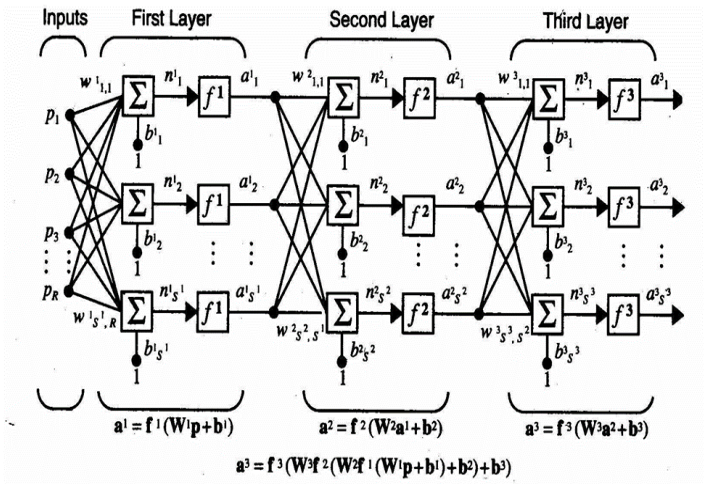

Figure 2. A multi-layer perceptron

We implemented a multi-layer perceptron network, with 7 output units. The first hidden layer consists of approximately double the number of input parameters. This number of hidden units is generally used in a multi-layer perceptron as a large number of hidden units lead to overfitting.

- Since there are 54 input parameters, the number of hidden units in the first hidden layer are 100.
- In the second hidden layer, we reduced this number to 10.
- The non-linearity function used in both layers was '**Tanh()**' due to its conveniently calculable derivative.
- Lastly, we used a **LogSoftMax** regression model to select the final activation unit [5]. This model generalizes logistic regression to classification problems where the class label *y* can take on more than two possible values (in this case, 7).
- While preprocessing the data, we normalized all columns to values between 0 and 1 by the following method.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

- For regularizing the cost function (a method used to counter overfitting), we augment the cost function with an additional regularization constant.
- The regularization parameter ƛ is a control on fitting parameters. As the magnitudes of the fitting parameters increase, there will be an increasing penalty on the cost function. This penalty is dependent on the squares of the parameters as well as the magnitude of ƛ.

VISION-SVHNPREDICTION

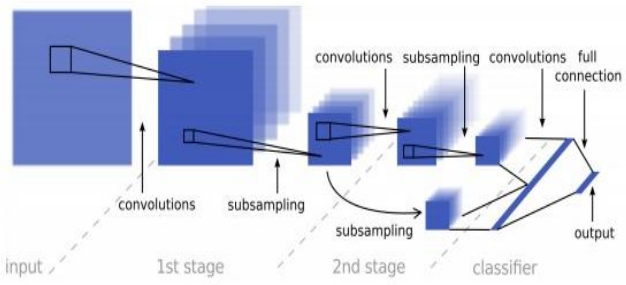

Figure 3. A 2 stage ConvNet architecture where Multi Stage features are fed to a 2 layer classifier. The 1st stage features are branched out, subsampled again, and then concatenated to 2nd stage features. [2]

Convolutional Neural Networks

ConvNets are hierarchical feature learning neural networks whose structure is biologically inspired. ConvNets learn features all the way from pixels to the classifier. Natural images have the property of being "stationary", meaning that the statistics of one part of the image are the same as any other part. This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations. More precisely, having learned features over small (say 8x8) patches sampled randomly from the larger image, we can then apply this learned 8x8 feature detector anywhere in the image. Specifically, we can take the learned 8x8 features and "convolve" them with the larger image, thus obtaining a different feature activation value at each location in the image.

Convolutions

Given some large $r \times c$ images x_{large}, we first train a sparse auto-encoder on small $a \times b$ patches x_{small} sampled from these images, learning k features $f = \sigma(W(1)x_{\text{small}} + b(1))$ (where σ is the sigmoid function), given by the weights $W(1)$ and biases $b(1)$ from the visible units to the hidden units. For every $a \times b$ patch x_s in the large image, we compute $f_s = \sigma(W(1)x_s + b(1))$, giving us $f_{\text{convolved}}$, a $k \times (r - a + 1) \times (c - b + 1)$ array of convolved features.

Pooling

After obtaining our convolved features as described earlier, we decide the size of the region, say $m \times n$ to pool our convolved features over. Then, we divide our convolved features into disjoint $m \times n$ regions. We may pool over this region such as max pooling or mean pooling. A specific kind of pooling, called L2-pooling, can be described by the following figure.

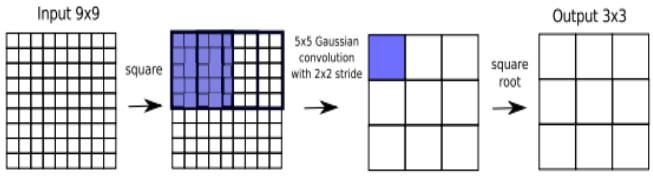

Figure 4. A diagram illustrating L2-pooling. [2].

ConvNets

A Convolutional Neural Network (CNN) is comprised of one or more convolutional layers (often with a subsampling step) and then followed by one or more fully connected layers as in a standard multilayer neural network. The architecture of a CNN is designed to take advantage of the 2D structure of an input image (or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features. Another benefit of CNNs is that they are easier to train and have many fewer parameters than fully connected networks with the same number of hidden units. A CNN consists of a number of convolutional and subsampling layers optionally followed by fully connected layers. The input to a convolutional layer is a $m\ x\ m\ x\ r$ image where m is the height and width of the image and r is the number of channels, e.g. an RGB image has $r = 3$. The convolutional layer will have k filters (or kernels) of size $n\ x\ n\ x\ q$ where n is smaller than the dimension of the image and q can either be the same as the number of channels r or smaller and may vary for each kernel. The size of the filters gives rise to the locally connected structure which are each convolved with the image to produce k feature maps of size $m - n + 1$. Each map is then subsampled typically with mean or max pooling over $p\ x\ p$ contiguous regions where p ranges between 2 for small images (e.g. MNIST) and is usually not more than 5 for larger inputs. We have used Lp-pooling as described in [2]. Either before or after the subsampling layer an additive bias and sigmoidal nonlinearity is applied to each feature map.

Data Pre-processing

The SVHN classification dataset [3] contains 32x32 images with 3 color channels. The dataset is divided into three subsets: train set, extra set and test set. The extra set is a large set of easy samples and train set is a smaller set of more difficult samples. This distribution allows to measure success on easy samples but puts more emphasis on difficult ones. Samples are pre-processed with a local contrast normalization (with a $7x7$ kernel) on the Y channel of the YUV space followed by a global contrast normalization over each channel.

Architecture

The ConvNet has 2 stages of feature extraction and a two-layer non-linear classifier. The first convolution layer produces 16 features with $5x5$ convolution filters while the second convolution layer outputs 512 features with $7x7$ filters. The output to the classifier also includes inputs from the first layer, which provides local features/motifs to reinforce the global features. The classifier is a 2-layer non-linear classifier with 20 hidden units. Hyper-parameters such as learning rate, regularization constant and learning rate decay were tuned on the validation set. We use stochastic gradient descent as our optimization method and shuffle our dataset after each training iteration. For the pooling layers, we compare Lp-pooling for the value p = 1, 2, 4, 8, 12.

EXPERIMENTS

Forest Cover Type Prediction

Our experiments with the network described above for this problem showed did not give good results on the test data. Supervised training of the MLP was performed. We experimented by changing the number of hidden layers, the number of hidden units in each layer, weight decay constant, batch sizes (in Stochastic Gradient Descent) etc. Our implementation used the ‘nn’ [7] neural network library in Torch7 for describing the neural network model. Next we used the ‘optim’ [8] library for running various optimization algorithms such as Stochastic Gradient Descent, ‘CG’, ‘LBFGS’, and ‘ASGD’, and also for determining the confusion matrix at each epoch.

SVHN

We used supervised training on the ConvNet. These deep learning neural networks are capable of learning multiple features of the input at each stage of learning. We used stochastic gradient descent as our optimization method and shuffled our dataset after each training iteration. In all there are about 73000 training images. We experimented with various values of ‘p’ in Lp-pooling. The architecture of the model has been described previously. There are 10 output classes. To implement the Convolutional Neural Network model and for running the optimization on the network, we use [7] and [8].

All experiments were done on the Torch7 platform, on Ubuntu 14.04 operating system using a standard

Dell (Intel Pentium i3 processor) laptop.

RESULTS AND DISCUSSIONS

Our experiments with a 2 layer neural network on the Kaggle challenge yielded poor results. For instance, at epoch number 11 of the training, the confusion matrix was the following.

Confusion Matrix:

Matrix	Row Accuracy	Class
[47 38 1 0 48 35 45]	21.963%	Spruce/Fir
[22 23 5 4 11 33 23]	19.008%	Lodge-pole Pine
[1 0 33 2 0 105 0]	23.404%	Ponderosa Pine
[0 0 0 0 0 0 0]	nan%	Cottonwood/Willow
[3 5 1 0 8 17 4]	21.053%	Aspen
[0 2 10 2 1 100 0]	86.957%	Douglas-fir
[79 45 1 0 43 10 113]	38.832%	Krummholz

Average row correct: 35.2026504774 89%
Average rowUcol correct (VOC measure): 19.1563335557 78%
Global correct: 35.217391304348%
Number of hidden units: Layer 1 ~ 100; Layer 2 ~ 10
Optimization method: Stochastic Gradient Descent
Number of test cases: 921

As seen, the predictions seem to be random and inaccurate. The multi-layer perceptron model is clearly not useful for learning in this case. On the other hand, Convolutional Neural Networks performed considerably well and fairly accurate predictions were made by it on the SVHN dataset. The following confusion matrix represents the results obtained for test data, a collection of about 23000 images, at the 4th training epoch.

Confusion Matrix:

Matrix	Row Accuracy	Class
[4782 20 38 114 11 19 51 13 3 48]	93.783%	1
[41 3855 54 51 31 27 38 21 21 10]	92.914%	2
[60 67 2355 36 109 59 9 102 76 9]	81.714%	3
[92 23 40 2256 27 30 17 8 15 15]	89.417%	4
[14 23 55 14 2098 145 5 10 15 5]	88.003%	5
[21 8 27 16 37 1790 6 35 5 32]	90.541%	6
[125 27 19 10 14 14 1793 2 8 7]	88.806%	7
[17 17 37 23 28 132 4 1352 26 24]	81.446%	8
[27 74 14 13 17 20 12 17 1330 71]	83.386%	9
[27 18 12 4 8 86 9 5 9 1566]	89.794%	0

Average row correct: 87.9804360866 55%
Average rowUcol correct (VOC measure): 78.7799030542 37%
Global correct: 89.032728948986%
Architecture as described in section 4

Clearly, we can see the superiority of ConvNets when dealing with a large number of parameters. The 'stationary' property of the images is exploited here, meaning that the statistics of one part of the image are the same as any other part. This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations i.e. 'convolve' them. The results in figure 6 are reported after 4 training epochs.

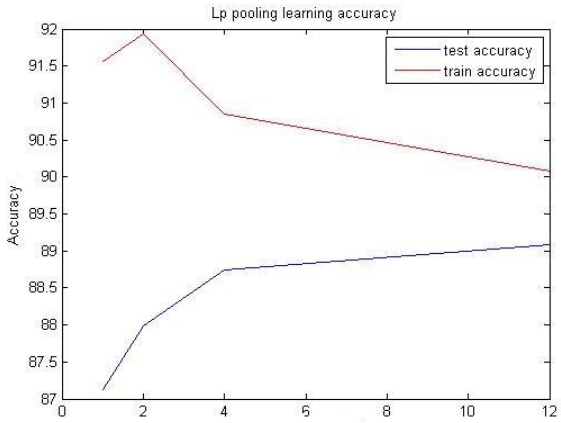

Figure 6. Training accuracy vs Lp-pooling

We believe these results can be improved upon greatly by employing greater hardware support for increased speed in training. It would also allow us to experiment with larger values of p and allow us to train for a much larger number of epochs. Results may also be improved by using unsupervised methods and we plan to do so in the future. From the results at hand, we can see that for p = 12, we achieve highest accuracy.

CONCLUSION

We conclude from our experiments and the results obtained that convolutional neural networks are extremely capable of learning a large number of features and the ability to capture different features at various levels is what makes them so useful, especially in the case of images. Conventional multi-layer perceptrons are inadequate by themselves to learn a large number of parameters as seen, and are prone to overfitting in such cases.

REFERENCES

[1] Bache, K. &Lichman, M. (2013). UCI Machine Learning Repository. Irvine, CA: University of

California, School of Information and Computer Science; http://www.kaggle.com/c/forest-cover-type-prediction
[2] Convolutional Neural Networks Applied to House Numbers Digit Classification. Pierre Sermanet, SoumithChintala and Yann LeCun, The Courant Institute of Mathematical sciences.
New York University.
http://arxiv.org/abs/1204.3968v1
[3] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*.(PDF);http://ufldl.stanford.edu/housenumbers/
[4] http://torch.ch/
[5]http://ufldl.stanford.edu/wiki/index.php?title=SoftmaxRegression&oldid=2288
[6]http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/
[7] https://github.com/torch/nn
[8] https://github.com/torch/optim