University of Paris Descartes, France
Master 1 computer science
Supervised research report project

Title

Deep neural networks and machine learning methods for image classification

Presented by : Mr. Ahmed MAZARI

Supervised by : Mr. Lazhar LABIOD and Mr. Mohamed NADIF
Examined by :

Mr. Laurent WENDLING
Mrs. Melissa AILEM
Mr. Ahmed MEHAOUA

Promotion 2015/2016

Abstract

Recently, deep learning models have drastically outperformed benchmark results on miscellaneous datasets. It's has become state-of-art in various domain especially computer vision and speech recognition. This breakthrough in artificial intelligence is associated with the availability of massive amount datasets and the high computing power. However, to train a deep learning model over a massive dataset makes the training slow and memory consuming. The target is to build models that generalize well on new examples in a reasonable time.
In this study, we address the problem of images classification by implementing different machine learning and deep learning algorithms. Then, we make a performance analysis on this methods under the accuracy criterion in order to show the efficiency and the contribution of deep learning. To do so, we've choosen linear and non-linear learning algorithms which are: Logistic Regression, Multi Layer Perceptron (MLP) , Support Vector Machine (SVM) and Convolutional Neural Networks (CNNs) tested on MNIST and CIFAR-10 datasets by applying the standard version of gradient descent optimization algorithm.

Keywords : deep learning, Convolutional neural network, supervised learning, kernels, optimization, hyperparameter, dimensionality reduction.

Contents

1 Introduction and related works **4**

2 Machine learning methods for image classification **5**
2.1 Reason for choosing these methods . . . 5
2.2 Logistic regression . . . 5
2.2.1 Implementation of logistic regression with gradient descent . . . 6
2.3 Multi Layer Perceptron . . . 6
2.3.1 Implementation of neural network with backpropagation . . . 7
2.3.2 Gradient checking . . . 8
2.4 Problem of parameter initialization . . . 9
2.5 Support vector machine . . . 9
2.6 Convolution neural network . . . 10

3 Datasets description and metric **11**
3.1 MNIST handwritten digits . . . 12
3.2 CIFAR10 . . . 12
3.3 Metric analysis : Accuracy . . . 13

4 Analysis and discussion of experimental results **13**

5 Choice of the learning algorithm **17**

6 Description of software and packages **18**

7 Conclusion **18**

8 References **19**

A
Code source **21**
A.1 Logistic Regression . . . 21
A.2 Multi layer Perceptron . . . 24
A.3 Support Vector Machine . . . 26
A.4 Convolutional Neural network . . . 27

List of Figures

1 Structure of deep neural network model . . . 4
2 Logistic regression with gradient descent pseudo code . . . 6
3 Structure of Multi Layer Perceptron-feedforward . . . 7
4 Structure of Multi Layer Perceptron-backpropagation . . . 8
5 Neural network with backpropagation pseudo code . . . 8
6 Example of SVM on vision problem . . . 10
7 Convolutional neural network . . . 11
8 MNIST dataset . . . 12
9 CIFAR-10 dataset . . . 13
10 Accuracy performance result with LogisticRegression and MLP on MNIST dataset . . . 14
11 Accuracy performance result with svm and cnn on MNIST dataset . . . 15
12 Benchmark results on MNIST dataset . . . 16
13 Benchmark results on CIFAR-10 dataset . . . 17
14 One-VS-All function Code . . . 21
15 Cost function for logisitic regression Code . . . 22
16 Sigmoid function . . . 23
17 Predict One-VS-All Code . . . 23
18 Multi Layer Perceptron Code Par1 . . . 24
19 Multi Layer Perceptron Code Part2 . . . 25
20 Multi Layer Perceptron Code Part3 . . . 26
21 SVM with polynomial Kernel Code . . . 26
22 Convolutional neural network main Code . . . 27

1 Introduction and related works

The rise of deep learning has drastically revolutionized the world of science in general and computing in particular. This sub-field of machine learning algorithms can be considered as a versatile approach that allows solving miscellaneous problems with a striking accuracy comparing to classical models. Recently, it has achieved the state-of performance in computer vision [27] : a visual object recognition task with 16 million images and 21000 categories.
The efficiency of deep learning comes from the computing power of calculators and the availability of massive datasets that allow deep models to learn complex features and capture regularities and invariants as we go more deeper in the shallow network. Deep models can significantly enhance performance and lead to a better accuracy.
Datasets keep growing dramatically to a billion of examples and millions of features, this is why we need complex (deep) models that support and take as inputs a high dimensional data and outputs a low-dimensional data so that they can be interpreted. It's still an open problem which is called « the curse of dimentionality » [25]. How is it possible to go from dimension N to dimension K such as $N >> K$ (Ex : 3D) ? What is the variable change required to transform these data ? [28].
Reasercher have been groping several ways to come up with a trick that reduce the dimenstionality of data. The recent achieved result is to train a restricted boltazman machine with raw data to reduce the dimension then train them with deep autoencoders [25] in the context of unsupervised features learning and Convolutional neural networks [6] for supervised data.
However, the memory requirement of deep learning models may exceed the memory capacity of a single CPU and GPU when it comes to visual data [29]. For instance to train DCNN (Deep Convolution Neural Network) with 1.2 million training images and 60 milion parameters using one GPU takes 10 days computing [30]. Moreover, deep learning models have to be trained several times to find optimal parameters that fit our data. Howerver, deep learning models are non-linear and non-convex which requires a block of operations and iteration to converge if initial coordinates are far from optimum. To tackle this issue, non-convex optimization routines should be defined such as SGD [31] (stochastic gradient descent) which can be executed in distributed asynchronous environment. Deep learning models take as input a high dimensional data and make blocs of complex mathematical iterations at each neuron of each layer. As far as we go deeper in the neural network we g build high level data representation. The structure of deep neural network is depicted in the following figure

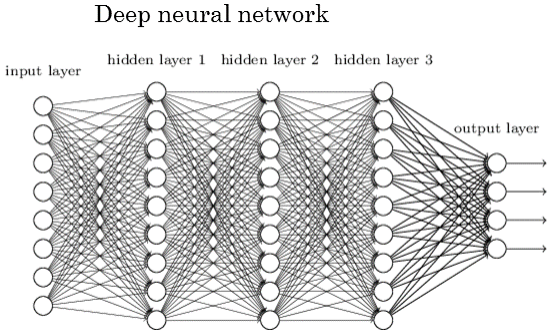

Figure 1: Structure of deep neural network model

In this project, we make a performance analysis study by comparing deep learning models to classical machine learning methods. We start by, implementing : logistic regression, multi layer perceptron, support vector machines then convolutional neural networks. Then we discuss the results using accuracy criterion in ordrer to consider the difference and the contribution of deep learning.

2 Machine learning methods for image classification

In our study, we focus on a set of supervised learning algorithms to analyze images. To do so, we put as a target to classify images according to their lablels. Classification can be defined as a method used to map inputs to outputs which are discrete categories. In classification problem the appropriate output to the input is already known. The challenge we address is to know wether the model can correcly classify new examples. From a huge family of supervised machine learning algorithms we have selected four learning algorithms which are:

- Logistic regression.
- Multi Layer Perceptron.
- Kernel methods.
- Convolutional neural networks.

To run these learning algorithms on a determined dataset, we need to associate them with an optimization algorithm so that to minimize the cost function.To do so, we have choosen to apply the standard version of gradient descent.

2.1 Reason for choosing these methods

The choice of these learning algorithms is not fortuitous. As the target is to make a comparative study on the efficiency of the aformentionned learning algorithms applied to images classification, we took different approaches : linear model, non linear model and convolutional model to come up with a conclusion avering the efficiency of each models according to its context.
From that point of view, each model has its own weaknesses and strenghts. They are intrinsically associated with the nature of data and the domain of application.

2.2 Logistic regression

Logistic regression is a linear supervised learning algorithm used to classify problems. It's a probablistic approach wherein classification is done by projecting input vector onto a set of hyperplanes and each hyperplane corresponds to a class. The distance between the input and the hyperplanes is the probability to affect the input to the corresponding class [1].
$Y = \{0, 1, 2.., n\}$ represents the set of classes wherein the example can be affected. Logistic regression can be considered as an improvement of multivariate linear regression. The limit of linear regression comes from the linearity of its hypothesis function : $h(\theta) = \theta_0 + \theta_1 x$.
This method doesn't work since most of classification problem are not linear. Formally multivariate linear regression can be presented as follows :

- **Hypothesis :**
  $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$ where $\theta, x \in R$
- **Learning parameters :**
  $\theta_0, \theta_1, \theta_2, ..., \theta_n$ where $\theta \in R$
- **Cost function:**
  $J(\theta_0, \theta_1, \theta_2, ..., \theta_n) = \frac{1}{2m}[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})^2 + \lambda \sum_{j=1}^{n} \theta_j^2]$
  where $\theta, h_\theta, y, \lambda$(**regularization parameter**) $\in R$
  the **target** is to minimize the cost function so that to converge to the global mininum :
  $\textbf{min}_{\theta_0, \theta_1, \theta_2, ..., \theta_n} J(\theta_0, \theta_1, \theta_2, ..., \theta_n)$

In our study we have implemented a "**one-vs-all**" method which is an improvement of linear regression. The hypothesis function h_θ in logistic regression is as follows : $0 \leq h_\theta \leq 1$ whereas in linear regression $0 > h_\theta$ or $h_\theta > 1$ even if data $\in [0, 1]$.
The modification of this hypothesis function allows to handle non linearities as the hypothesis function becomes :
$Max_i\ h_\theta(x)^{(i)}$= $P(Y = i|x, \theta)$.
$h_\theta(x) = g(\theta^T x)$ and $g(\theta^T x) = \frac{1}{1+\exp^{-\theta^T x}}$ which is called '"sigmoid function".
The sigmoid function is a $non - convex$ function , so it can't be applied to the previous cost function beacause when we run $J(\theta)$ there is no guarantee of convergence to the global minimum, this is why we should transform our cost function to get a convex function.
The cost function $J(\theta)$ is expressed as follows :
$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(h_\theta(x^{(i)})) + (1 - y^{(i)})log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2.$

2.2.1 Implementation of logistic regression with gradient descent

In our study we have implemented the basic gradient descent. The aim of target descent it to minimize the cost function $J(\theta)$. To do so, we first calculate the partiel derivative of the cost function $J(\theta)$ as follows :
$\frac{\delta}{\delta\theta_j}J(\theta) = \frac{\delta}{\delta\theta_i}\frac{-1}{m}\sum_{i=1}^{m}[y^{(i)}log(h_\theta(x)^{(i)}) + (1 - y^{(i)})log(1 - h_\theta(x^{(i)}))] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$

Here is the pseudo code implementation of logistic regression with gradient descent :

Function LogisticRegression()

Init: $i \leftarrow 1; j \leftarrow 1;$

Repeat

$\theta_j \leftarrow \theta_j - \frac{\alpha}{m}[\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)} + \lambda\theta_j]$

Until convergence

Figure 2: Logistic regression with gradient descent pseudo code

2.3 Multi Layer Perceptron

Multi layer Perceptron (MLP) is one of the non linear supervised learning algorithm. MLP can tackle complex datasets that linear algorithms fail to build complex models that fit the data. The first layer is called input layer that contains data from which the algorithm learn. The last layer is called output layer which givs the final value(s) computed on the hypothesis. If the model don't make mistakes the output values correspond to the labels. Between, input layer and output layer we can find a set of layers called hidden layers. Each hidden layer contains a number hidden units (neurons). We define the different symbols that describe the variables used to form the cost function of our neural network.
$a_i^{(j)}$ = activation of unit i in layer j
$\theta^{(j)}$ = matrix of weights cotrolling function mapping from layer j to layer j+1
$g(\theta^T x) = \frac{1}{1+\exp^{-\theta^T x}}$ wich is a sigmoid function
$z_k^{(j)} = \theta_{k,0}^{(1)}x_0 + \theta_{k,1}^{(1)}x_1 + .. + \theta_{k,n}^{(n)}x_n$ j : the layer, k : the node within a layer j.
$z^j = \theta^{(j-1)}a^{(j-1)}$
$a^j = g(z^{(j)})$
L = total number of layers in the network.
s_l = number of units(neurons) without counting bias unit in layer l.

$K =$ number of output classes.
for example : $a_1^{(2)} = g(\Theta_{10}^{(1)}x_0 + \Theta_{11}^{(1)}x_1 + \Theta_{12}^{(1)}x_2 + \Theta_{13}^{(1)}x_3)$
To define the cost function for a neural network, we can consider logistic regression. An MLP can be viewed as a logistic regression where the inputs are transformed using a learnt non-linear transformation θ. The idea behind this non-linear transformation is to project data into a space where it becomes linearly separable. The cost function is slightly different from the one of logistic regression. It's defined as follows :

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}\sum_{k=1}^{K}[y_k^{(i)}log((h_{\theta(x^{(i)})})_k) + (1-y_k^{(i)})log((1-h_{\theta(x^{(i)})})_k)] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2$$

We notice that the cost function of neural network is just an improvement of logistic regression. The double sums adds up the logistic regression costs calculated for each cell in the output layer and the triple sum adds up the square of each Θ_s in the entire network. The following Figure depicts the structure of the standard multi layer perceptron [2].

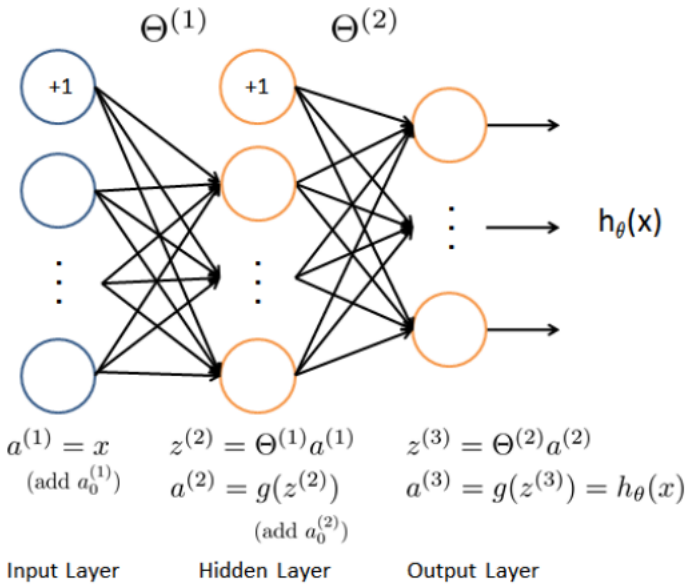

Figure 3: Structure of Multi Layer Perceptron-feedforward

2.3.1 Implementation of neural network with backpropagation

In the area of neural network the terminology used to define optimization function is called **backpropagation**. It's governed by the same property than gradient descent with logistic regression. The target is to compute $min_\theta J(\Theta)$, so we compute the partial derivative of $J(\Theta)$:
$\frac{\partial}{\partial\theta_{i,j}^{l}}J(\theta)$. For each node we compute :
Δ_j^l : error of node j in layer l.
The delta value for the last layer is easy to compute. It's the differences of the actual results in the last layer and the correct outputs in y.
$\Delta^{(L)} = a^{(L)} - y$.
However, the Δ values of layer l are much complicated to compute. They are calculated by multiplying the delta values in the next layer with the Θ matrix of layer l. Then, we multiply that with a function called $g^{'}$ which is the derivative of the activation function g evaluated with the input values given by $z^{(l)}$:
$\Delta^{(l)} = ((\Theta^{(l)})^T\delta^{(l+1)}) .* g^{(')}(z^{(l)})$
such that : $g^{(')}(z^{(l)} = a^{(l)} .* (1 - a^{(l)})$

$g^{'}(z^{(l)}) = \frac{\partial_g(z)}{\partial_z} = -(\frac{1}{1+\exp^{-z}})^2 \frac{\partial}{\partial z}(1 + exp^{-z}) = -(\frac{1}{1+\exp^{-z}})^2 exp^{-z}(-1) = (\frac{1}{1+\exp^{-z}})(\frac{1}{1+\exp^{-z}})(\exp^{-z}) = (\frac{1}{1+\exp^{-z}})(\frac{\exp^{-z}}{1+\exp^{-z}}) = g(z)(1-g(z))$
Finally, our partial derivative terms becomes :
$\frac{\partial J(\Theta)}{\partial \Theta^l_{i,j}} = \frac{1}{m}\sum_{t=1}^{m} a_j^{(t)(l)} \delta_i^{(t)(l+1)} + \frac{\lambda}{m}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}} \Theta_{j,i}$
The backpropagation architecture is illustrated in the following Figure [2].

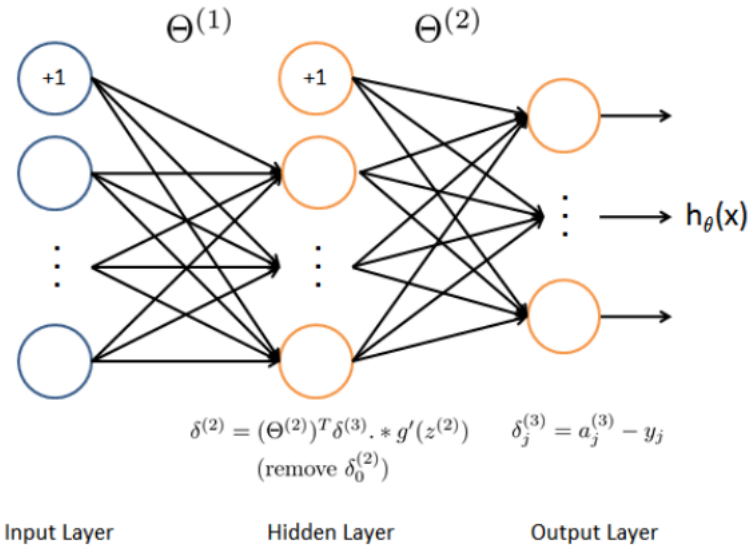

Figure 4: Structure of Multi Layer Perceptron-backpropagation

Here is the pseudo code Implementation of a neural network with backpropagation that represents the optimization process :

Function backpropagations()

A- Given training set $(x^{(1)}), y^{(1)} ... (x^{(m)}), y^{(m)}$
B-Set $\Delta_{i,j}^{(l)} := 0$ for all (l,i,j)
C-For training example $t = 1$ to m :
1-Set $a^{(1)} := x^{(t)}$
2 Perfom forward propagation to compute $a^(l)$ for $l = 2, 3, ..., L$
3-Using y^t, compute $\delta^{(L)} = a^{(L)} - y^{(t)}$
4-Compute $\delta^{(L-1)}, \delta^{(L-2)}, .., \delta^{(2)}$
D- Cost :
1-$D_{i,j}^{(l)} := \frac{1}{m}\Delta_{i,j}^{(l)}$ if $j = 0$
2-$D_{i,j}^{(l)} := \frac{1}{m}(\Delta_{i,j}^{(l)} + \lambda\Theta_{i,j}^{(l)})$ if $j \neq O$

Figure 5: Neural network with backpropagation pseudo code

2.3.2 Gradient checking

In order to check whether our backpropagation algorithm works well, we approximate the partial derivative of the cost function as follows :

$\frac{\partial}{\partial \Theta_j} J(\Theta) \approx \frac{J(\Theta_1,...,\Theta_{j+\epsilon},...,\Theta_n) - J(\Theta_1,...,\Theta_{j-\epsilon},...,\Theta_n)}{2\epsilon}$

It's important to choose the appropriate value of ϵ to avoid numerical problems. Usually, we use $\epsilon = 10^{-4}$ as it's suggested in [2] . After that, we compare between gradient checking and Delta. it should guarantees the following :

$$GradientChecking \approx D_{i,j}^{(l)} := \frac{1}{m}(\Delta_{i,j}^{(l)} + \lambda \Theta_{i,j}^{(l)})$$

2.4 Problem of parameter initialization

Initializing the weights, hyperparameter and ϵ is a difficult problem. The first idea that comes to mind is to initialize all the weights to zeros. Unfortunately, it's doesn't work because this way of initialization gives rise to the same update value at each node repeatedly at each iteration. To circumvent this issue, we can instead initialize the weights randomly :

1. $\epsilon = \frac{\sqrt{6}}{\sqrt{Layeroutput + Layerinput}}$ $Layeroutput(input)$: the dimension of the output (input) layer.
2. $\Theta^{(l)} = 2 * \epsilon * rand(Layeroutput, Layerinput + 1) - \epsilon$

2.5 Support vector machine

Support Vector Machine (SVM) is another family of supervised learning algorithm as logisic regression and neural networks. It's the main kernel algorithm that can be defined as a similarity measure [4]. The objective of SVM comparing to the aformentionned methods is to build optimal hyperplanes in a multidimensional space, which can separate data from opposite classes using a distance called $margin$. Margin is a distance between the optimal hyperplane and a vector which lies closest to it.

To tackle a non-linear separability problem we have a a variant of kernels such as linear kernel, polynomial kernel, gaussian radial kernel and gaussian radial basis polynomial kernel [3].

To make a support vector machine, we modify a bit the cost function of logisic regression (hypothesis and the regularization parameter).

$-log(\frac{1}{1+\exp^{-\Theta^T x}})$ becomes $-log(h_\Theta(x)$ and

$-log(1 - \frac{1}{1+\exp^{-\Theta^T x}})$ becomes $-log(1 - h_\Theta(x))$.

We denote $cost_1(z)$ and $cost_0(z)$ respectively the cost for classifying when $y = 1$ and when $y = 0$.

$cost_0(z) = max(0, k(1 + \Theta^T x))$ such that k is a constant which defines the magnitude of the slope of the line.

$cost_1(z) = max(0, k(1 - \Theta^T x))$

We multiply our cost function by m and set a regularization parameter $C = \frac{1}{\lambda}$

$$J(\Theta) = C \sum_{i=1}^{m} y^{(i)} cost_1(\Theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\Theta^T x^{(i)}) + \frac{1}{2} \sum_{j=1}^{n} \Theta_j^2$$

For instance, SVM separates the negative and positive examples by a large margin which can be achieved when C is very large (λ very small). So, we keep increasing and decreasing C to fit the model to the data.

An SVM is defined using a kernel to make complex and non-linear classifier. We have choosen 3 kernels which are :

1. **Linear Kernel** : $K(x, x') = x.x'$
2. **Polynomial Kernel** $K(x, x') = (1 + x.x')^d$ where d is the degree of polynomial function.
3. **Gaussian Radial Kernel** $f_i = similarity(x, l^{(i)}) = \exp(-\frac{\sum_{j=1}^{n}(x_j - l_j^{(i)})^2}{2\sigma^2})$
   $l^{(i)}$ represents a landmark near to a given point x.
   f_i is a Gaussian Kernel which computes the similarity between x and $l^{(i)}$ [4].

To do a multi-class classification, we proceed as we did with logisic regression using one-vs-all method and take class i with the largest $(\Theta^{(i)})^T x$. The following Figure illustrates an example of multi class classification of vision problem using SVM [26]

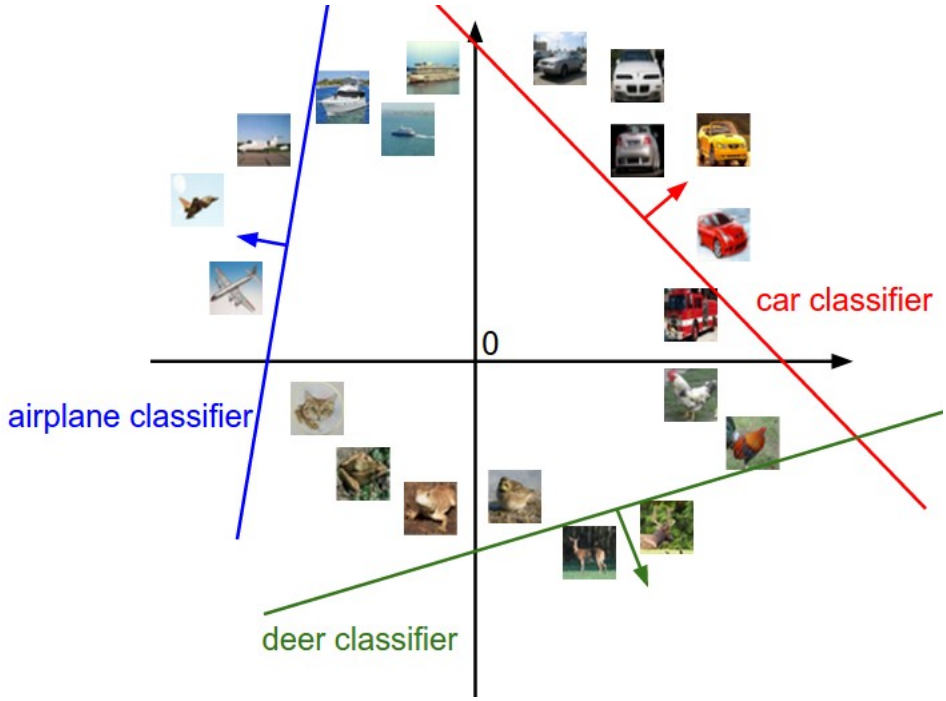

Figure 6: Example of SVM on vision problem

2.6 Convolution neural network

Convolution Neural Networks (CNN) are a variant of Multi Layer Perceptron (MLP). To do tasks as object, sound, text recognition, CNN tries to mimic human brain. For instance, tasks related to vision, CNN tend to imitate the visual cortex of human brain [5].
CNN differs from MLP in the way that each hidden unit of layer $l^{(i)}$ receives weights from all units of layer $l^{(i-1)}$. However, in CNN each hidden unit represents a receptive field that acts like a filter according to its own function. The function varies from a receptive field to another. Each receptive field receives specific weights from layer $l^{(i-1)}$ (not all the weights of $l^{(i-1)}$). In other words, each hidden unit in layer $l^{(i)}$ composed of a subset of units in layer $l^{(i-1)}$. These receptive fields are unresponsive to variations outside of its receptive field [6].
We note that these filters (receptive fields) are locally invariants. However, we want to have filters which are globally invariants. To do so, we stack many layers with these kind of filters that have non-linear properties so that to guarantee global invariance.
The main idea behind CNN is the replications of units at each layer and weights sharing. This trick allows to detect features regardless of their position in the visual field. These replications are done using convolution which enables to get feature map by repeated application of a function across sub-regions of the entire image. Set these two constaints (units replication and weights sharing) to the network enables CNN to achieve better generalization on vision problems.
The convolutional operation applied here is as follows : each output feature map is connected to each input feature map by a different 2D filter, and its value is the sum of the individual convolution of all inputs through the corresponding filter [7].
We denote the k-th feature map at a given layer h^k, weights W^k, bias b_k and x the input image. We determine h^k by W^k and b_k applying non-linear funcion $\tanh$ where each hidden layer is composed of multiple feature maps.

$$h_{ij}^k = \tanh((W^k * x)_{ij} + b_k).$$

The weights W of a hidden layer is a combination of destination feature map, source feature map, source vertical position and source horizontal position. The bias can be represented as one element vector that maps every destination. The convolution operator used in CNN are 4D Tensors. Tensors are geometric objects that describe linear relations between geometric vectors, scalars, dot product, cross product and linear maps [8]. The 4D tensor corresponds to the input images and to the weight matrix W. The tensor for the input images is composed of mini-batch size, number of inupt, feature maps image width and height. The tensor corresponding to the weight matrix W consists of a number of feature maps at layer $l^{(i)}$, number of feature maps at layer $l^{(i-1)}$, filter width and height. We get the weights randomly as in MLP following a uniform distribution in the range $[-\frac{1}{hd-in}, \frac{1}{hd-in}]$ where $hd - in$ is the number of inputs to a hidden unit.

The main difference with MLP is in term of the structure of relationship between units. In MLP, the number of inputs to a hidden unit is determined by the number of units in the layer below. However, in CNN the input to a hidden units intrinsically depends on the feature maps and the size of the receptive field.

Convolution layer is followed by a MaxPooling layer. It partitions the input image into a set of non-overlapping rectangles. Then, for each sub-region, it outputs the maximum value [6]. The contribution of MaxPooling [9] to CNN is in term of dimensionality reduction. From one hand, it reduces computation for upper layers because it takes the maximum value of each sub-region and eliminate the remained values. From other hand, it makes some translation from which we get invariance. The following Figure illustrates the general structure of Convolutional Neural Network.

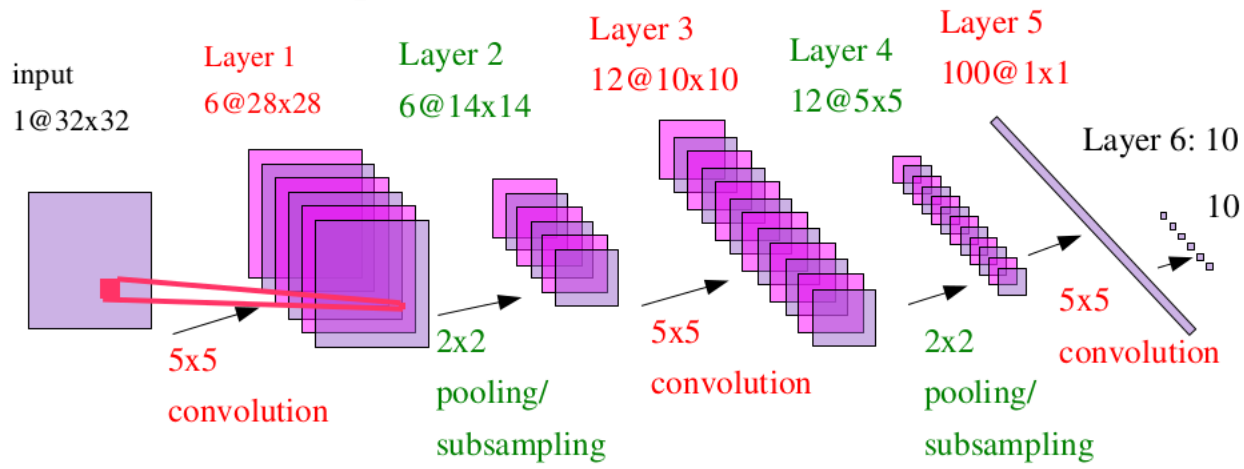

Figure 7: Convolutional neural network

The three most striking ideas that we can hold back are : *equivariant representations* got from translation (MaxPooling), *parameter sharing* (Receptive field) and *sparse interactions*. Sparse interactions is achieved by making the kernel smaller than the image input. This sparse interaction drive us also to dimensionality reduction where only meaningful features are kept.

3 Datasets description and metric

In this section, we give an overview of two datasets we've used during our experimental analysis. We first intrdouce MNIST dataset [10] which is a handwritten digits database. Then, we describe CIFAR-10 dataset [11] which is a database of natural images.

Since our study is to make a performance analysis of different learning algorithms, it's fundamental to associate them

with a metric which evaluates the efficiency of each learning algorithm.
From several metrics : $Recall$, $F_1 score$, $Accuracy$, $Confusion\ matrix$, we have choosen $accuracy$ criteria to evaluate Logistic Regression, MLP, SVMs and CNN on MNIST [10] and CIFAR-10 [11].

3.1 MNIST handwritten digits

The MNIST dataset [10] is a handwritten digit images. It contains $60,000$ examples for the training set and $10,000$ examples for testing set. It's collected by Yann LeCun, Corinna Cortes and Christopher J.C. Burges. All digit images have been size-normalized and centred in a fixed size image of 28x28 pixels. Each pixel of the image is represented by a value between 0 and 255, where 0 is black, 255 is white and anything in between is a different shade of grey [7].
Here are some examples of MNIST dataset :

Figure 8: MNIST dataset

3.2 CIFAR10

The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are $50,000$ training images and $10,000$ test images. The dataset is divided into five training batches and one test batch, each with $10,000$ images. The test batch contains exactly 1000 randomly-selected images from each class. The training batches contain the remaining images in random order, but some training batches may contain more images from one class than another. Between them, the training batches contain exactly 5000 images from each class. It was collected by Alex Krizhevsky, Vinod Nair and Geoffrey Hinton [11].
We have 10 classes. Each class contains 10 different images. The classes are : airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The classes are depicted in the following figure :

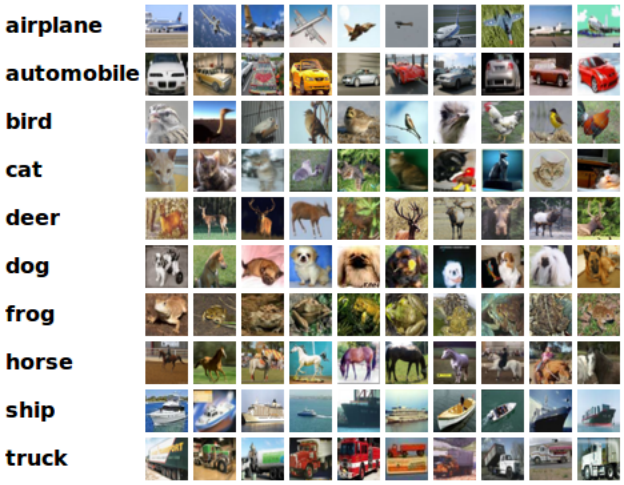

Figure 9: CIFAR-10 dataset

3.3 Metric analysis : Accuracy

Accuracy is a metric that evaluates the performance of a given classifier. It measures the number of times the classifier makes the correct prediction.
$Accuracy = \frac{\#correctpredictions}{\#totaldata}$ such that :
#correct prediction : the total number of correct prediction.
#total data : the total number of data.

4 Analysis and discussion of experimental results

We've evaluated the following learning algorithms : logistic regression, MLP, SVMs, CNNs by applying them to handwritten digit image recognition [10]. Then, we commented some benchmark results on CIFAR-10 natural images dataset using the recent neural learning technics such as fractional max pooling (FMP) and Multi column deep neural network (MCDNN). We first implemented logistic regression with 20, 25, 100, 200, 500,1000 then 2000 gradient iterations. We've reached 96.46% accuracy with 500 iterations . The result remains the same with 1000 and 2000 gradient iterations. Then, we deduce that the model becomes stable after 500 gradient iterations. Since the accuracy results remains the same with 2000 iterations we keep 500 as the gradient parameter of our model.
After that, we implemented a multi layer Perceptron (MLP) with 3 layers : input layer, 1 hidden layer with 25 neurons (units) and output layer with 10 neurons whic represents the lables $[0..9]$ applying 20, 25, 100, 200, 500 then 1000 gradient iterations. We've reached 92.85% accuracy with 1000 iterations .
From a practical standpoint, we noticed that logisic regression with 100 gradient iterations is more efficient (95.9% accuracy) than MLP of 1 hidden layer with 25 neurons and 1000 gradient iterations (92.85%).
In order to refine our study, we added 275 neurons to the previous MLP to have 300 hidden units. The results obtained with this new architecture outperform those of logistic regression and MLP (25 hidden units) even with only 20 gradient iterations. We got 96.28% accuracy with 20 gradient iterations and 97.91% accuracy with 100 gradient iterations where neither logistic regression nor MLP (25 hidden units) with 1000 gradient iterations achieved this result. The only constraint is that large and shallow MLP are slow to train, time and memory consuming.
Moreover, we made the architecture of MLP more complex by stacking 2 and 3 hidden layers. The first MLP with 2 hidden layers contains respectively of 300, 150 hidden neurons. The second MLP with 3 hidden layers contains respectively 300, 150, 75 hidden neurons. The training was very slow. The MLP with 2 hidden layers gave 10.05% accuracy with 100 gradient iterations. The MLP with 3 hidden layers gave 11.35% accuracy with 100 gradient itera-

tions.
We noticed that the simple MLP (300 hidden units) outperforms drastically the shallow MLP (2 hidden layers). From that observation, we conclude that the more the network is deep doesn't mean it gives better performance. The number of hidden units and hidden layers depends on the model and data type [12]. Our results on MNIST applying Logistic Regression and MLPs are depicted in the following figure.

Gradient iterations / Learning algorithms	Accuracy with different number of gradient iterations					
	20	25	100	200	500	1000
Logistic Regression			95.9%	96.32%	**96.46%**	96.46%
Multi layer perceptron with 25 hidden neurons	89.14%	90.14%	91.05%	92.02%	92.74%	**92.85%**
Multi layer perceptron 300 hidden neurons	96.28%	96.62%	98%	98.12%	98.21%	**98.22%**
Multi layer perceptron 300-150 hidden neurons			**10.05%**			
Multi layer perceptron 300-150-75 hidden neurons			**11.35%**			

Figure 10: Accuracy performance result with LogisticRegression and MLP on MNIST dataset

The values in bold blue color represent our best performance achieved. However, the bold red ones represent our worst result obtained.
To tackle the concern of appropriate number of hidden layers and units, we come back later on to shallow architecture : Deep Convolutional Neural Network (DCNN) to evaluate its performance on MNIST dataset.
Furthermore, we implemented Support Vector Machine (SVM) with linear, polynomial and Gaussian radial kernel. Neural networks and SVMs both give promising results. It's difficult to predict which one fit better the data before doing some experimentations. We got 93.98% accuracy with linear kernel, 98.08% accuracy with polynomial kernel and 94.46% accuracy with gaussian radial kernel.
From these results, we've noticed that SVM with polynomial kernel is as efficient as MLP with 300 hidden units (more than 100 gradient iterations). This accuracy performance on MLP and SVM open an eye to critical analysis before choosing any model for data. We come back in the next section to give some analytical elements to take into consideration to choose between logisic regression, MLPs and SVMs.
Finally, We arrive to the latest learning algorithm intended for image recognition. Recently, Convolutional Neural Networks achieved state-of-art performance on MNIST and CIFAR-10 datasets [13]. We have trained 2 Convolutional neural networks of 2 Convolutionals layer and two subsampling layers respectively with 3 and 10 epochs. The results were almost equals. We got 95.11% accuracy with 5 epochs and 95.66% with 10 epochs.
Our results using SVMs and CNNs are illustrated in the following figure :

Learning algorithms \ Accuracy		**Accuracy**
Support Vector Machine (SVM)	Linear Kernel	93.98%
	Polynomial Kernel	**98.08%**
	Gaussian Radial Kernel	94.46
Convolutional neural network (CNN)	5 epochs with two hidden layers	95.11%
	10 epochs with two hidden layers	**95.66%**

Figure 11: Accuracy performance result with svm and cnn on MNIST dataset

An epoch is a bloc of iterations. For example, if we train our learning algorithm with 5 epochs and we set 500 as a gradient parameter, it means that the learning algorithm will be executed five time each with 500 gradient iterations. Comparing the performance of Logistic Regression, MLPs, SVMs and CNNs, we notice that MLP with 300 hidden units gives better result 98.22% accuracy applying 500 gradient iterations.
However, it's too early to extrapolate on the best learning algorithm for image classification. For that reason, we've examined some recent results on CIFAR-10 dataset [11] where the latest learning technics were applied.
Before diving into the analysis of benchmark results [13], l come up with a set of fundamental questions to ask before any decision making upon the model to adopt :

- Is there any learning algorithm betther than the other without taking into consideration context and type of data?
- Are empirical results more trusty than theoritical analysis of each approach ?
- How to choose the number of hidden layers and units ?
- How well to choose the optimization algorithm that fits the choosen model ?
- What's the trade-off between time learning and accuracy performance we are looking for ?
- Does the choosen model best generalize on new examples ?

Multi-column Deep Neural Network (MCDNN) [14] and DropConnect [15] have drastically imporved performance accuracy on MNIST dataset [10]. DropConnect [15] achieved state-of-art performance on MNIST dataset [10] in 2013 with 99.79% accuracy followed by MCDNN [14] in 2012 with 99.77%. The results are depicted in the following figure :

Learning algorithms \ Accuracy		**Accuracy**
Multi Layer Perceptron (MLP) [13]	4 hidden layers (2000-1500-1000-500) with **5% translation**	99.06%
	5 hidden layers (2500-2000-1500-1000-500)	98.53% after 14 epochs
Multi-Column Deep Neural Network (MCDNN) [14]		99.77%
Neural network with DropConnected [15]		**99.79%**

Figure 12: Benchmark results on MNIST dataset

Transaltion means that the handwritten digit is moved or cover some distance. For instance, if we take any handwritten digit which is located on the right side of the picture. If the handwritten digit is moved to the left side of the picture then the process is called transaltion.
Multi-Column Deep neural network is a technic where the input image can be preproccessed by $P_0..P_{n-1}$ blocks. An arbitary number of columns can be trained on inputs preprocessed in different ways. The final predictions are obtained by averaging individual predictions of each DNN [14].
DropConnect is a regularization technic where each connection can be dropped with probability $1 - p$ [15]. It introduces dynamic sparsity within the model on the weights W rather than the output vectors of a layer. It's the improvement of DropOut technic suggested by Geoffrey Hinton [16].
Considering these benchmark results, we notice that MLP with 4 hidden layer (99.06%) gave a better result than MLP with 5 hidden layers (98.53%). We confirm what we've just said before. Training a model with a shallow architecture doesn't mean achieving better performance. In fact, The MLP with 4 hidden layers gave better result thanks to transaltion. It's important not to stack hidden layer blindly. From this empirical result we've learned that a network with few hidden layer composed of some transformation and hierarchical structure gives better results. It's the case with Convolutional Neural networks [15, 16] where functions are structured and each hidden unit has its own structure. hidden units don't present a simple neuron like in MLP but can be a filter, a rectifier, a function of transaltion or rotation. This architecture is less sensitive to noise. However, MLPs Don't allow to make transaltion, rotation and they are sensitive to noise and distortion. This is why our results using MLP with 2 and 3 hidden layers illustrated in **Figure 6** are so poor. The inner structure of hidden layers is the same where every hidden neuron of each hidden layer does the same thing. In contrast, CNN are well structured to learn invariants from transaltion beacause the hidden units present different experts and each expert has different functions.
The reason why our MLP with 2 and 3 hidden layers gave a poor results as depicted in **Figure 6**, is that building these complex models, it's quite easy to perfeclty fit our MNIST dataset [10]. We computed the accuracy on the training set then we got 99.66% accuracy. But when we evaluate such a complex model on new data, it performs very poorly. Which means that our model does not generalize well. It's called overfitting. According to the results illustrated in **Figure 6**, MLP of 1 hidden layer with 300 hidden neurons is sufficient to best classify MNIST dataset.
However, MLP and Logistic Regression can't be applied to complex data such as object recognition in images : face, nose, ears, eyes, mountains and object in motion ... Because their inner structure don't take into consideration transaltion, rotation, distorsion and scaling. However, Convolutional neural network [6] are better suited to image recognition. Besides, it's know possible to tackle this problem with kernels associated to deep learning [17]. To better illustrate that, we describe in the following figure bechmark results on CIFAR-10 [11].

Accuracy / Learning algorithms	Accuracy on CIFAR-10
Fractional max-pooling [18]	**96.53 %**
Simple CNN [19]	95.59 %
Init CNN [20]	94.16%
MCDNN [14]	88.79%

Figure 13: Benchmark results on CIFAR-10 dataset

Fractional max-pooling [18] is the stochastic version of max-pooling where α is allowed to take non-integer values. This form of max-pooling reduces overfitting. However, the standard form of max-pooling is αxα with $\alpha = 2$. The result obtained represents State-of-Art on CIFAR-10 dataset [11].
The idea of striding within a convolutional layer is tackled in [19]. It means that the convolutional layer uses a filter of size 2x2 and slides it over the entire image and each time it stops, the filter performs a funcion involving the pixels that it lies on.
In [20] Layer-sequential unit variance (LSUV) initialization is introduced. The idea behind this initialization consists of pre-initializing weights of each convolution layer and proceeding to variance normalization of the output of each layer to be equal one.

5 Choice of the learning algorithm

Coming up with the appropriate learning algorithm that fits our data is a difficult problem. Several approaches are suggested in the literature where each approach has its own advantages and disadvantages.
The choice of the method is intrinsically related to the structure of data. let's m the number of examples and n the number of features. To choose between SVMs and Logistic Regression, Andrew ng [2] suggested the following :

- If n is large (relative to m), then use logistic regression, or SVM without a kernel (the "linear kernel").
- If n is small and m is intermediate, then use SVM with a Gaussian Kernel
- If n is small and m is large, then manually create/add more features, then use logistic regression or SVM without a kernel.

In the first case, we don't have enough examples to need a complicated polynomial hypothesis. In the second example, we have enough examples that we may need a complex non-linear hypothesis. In the last case, we want to increase our features so that logistic regression becomes applicable.
The problem of choosing the appropriate kernel (linear, polynomial, gaussian radial) is also to consider. The first idea that came to our mind is Cross validatio. It means try several different Kernels and evaluate their output-of-sample performance.
According to **No-Free Lunch Theorem** [21] there are no guarantees for one kernel to work better than the other. From a practical point of view [22, 23] regularization technics are suggested to choose the appropriate kernel for our data.
Recently, SVMs and Neural Network are combined [17]. It is likely to have better classification results compared to standard neural network. for instance, we might utilize many layers of neurons and have the final classification via SVM athe the output layer.
Neural network are good if we have many training examples and we can tune our learning and regularization parameter. In contrast, it's better to use SVMs if we don't have many training examples because SVM training is quadratic in the number of examples. SVMs have a strong foundation theory, they reach global optimum thanks to quadratic complexity and they are also less prone to overfitting since we don't have to choose a certain number of parameters comparing to neural networks (non-convex optimization).From a research point of view both SVMs and Neural nets

are black boxes.

6 Description of software and packages

We did our experiments on MATLAB 2015a. We used DeepLearning toolbox and libsvm library.

1. **Deep Learning toolbox** contains a set of library for deep learning. We used the library CNN for Convolutional neural networks. It has also libraries for feedforward backpropagation Neural Networks (NN), Deep Belief Networks (DBN), Stacked Auto-Encoders (SAE), Convolutional Auto-Encoders (CAE).

2. **Libsvm** is a library for Support Vector Machines. It's a very fast software and easy to use.

7 Conclusion

This supervised research project allowed us to better understand the latest approaches of deep neural network from theoretical and practical standpoints. We've learned the different machine learning and deep learning models and their domains of application. We've tackled the problem of choosing the appropriate model for a given data. Besides, We noticed how different is image processing from other type of data.
As a perspective, we intend to explore deep learning for clustering and dimensionality reduction from practical and mathematical point of view, using restricted boltzman machine [24] and stacked Auto-Encoders [25]. Then, we cope with the non-convex optimization, number of hidden layers and units so that to built optimized models that generalize well on new examples.

8 References

References

[1] Lisa Lab, deep learning tutorial. University of Montreal

[2] Andrew ng.Machine learning class. Online classroom, Stanford University

[3] E.A. Zanaty.Support Vector Machines (SVMs) versus Mutlilayer Perceptron (MLP) in data classification. Volume 13, Issue 3, pages 177-183, Egyptian Informatics Journal. November 2012

[4] Bernhard Schölkopf and Alexander J.Smola.Learning with kernels : Support Vector Machines, regularization, Optimization and beyond.

[5] Hubel, D. and Wiesel, T.(1968). Receptive fields and functional architecture of monkey striate cortex. Journal of Physiology (London), 195, 215-243.

[6] LeCun, Y., Boutto, L., Bengio, Y., and Haffner, P. (1998d). Gradient-based learning applied to document recognition.

[7] Deep learning tutorial website. www.deeplearning.net

[8] Wikipedia. Tensor

[9] Zhou and Chellapa 1988.

[10] http://yann.lecun.com/exdb/mnist/

[11] https://www.cs.toronto.edu/ kriz/cifar.html

[12] Mac Mèzard and Jean-Pierre Nadal. Learning in feedforward layered networks : the tiling algorithm. J. Phys. A: Math. Gen. 22 (1989) 2191-2203.

[13] Dan Claudiu Ciresan. Benchmark on images classification datasets. Senior Researcher, Dalle Molle Institute for Artificial Intelligence , Swissterland

[14] Dan Ciresan, Ueli Meier and Jürgen Schmidhuber. Multi-column Deep Neural Networks for Image Classification. cvpr2012

[15] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Lecun and Rob Fergus. Regularization of Neural Networks using DropConnect. ICML 2013

[16] G.E.Hinton, N. Srivastava, A.Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. CoRR, abs/1207.0580, 2012.

[17] Youngmin Cho and Lawrence K.Saul. Kernel methods for Deep Learning. NIPS 2009

[18] Benjamin Graham. Fractional Max-Pooling. May 2015

[19] Jost Tobias Springenberg, Alexey Dosovitskiy, Thomas Brox, Martin Riedmiller. Striving for simplicity : the all convolutional net. ICLR 2015.

[20] Dmytro Mishkin and Jiri Matas.All you need is a good init. ICLR 2016.

[21] David H. Wolpert and William G. Macready. No Free Lunch Theorem for Optimization. IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, Vol. 1, NO. 1, APRIL 1997

[22] G. C. Cawley and N. L. C. Talbot, Preventing over-fitting in model selection via Bayesian regularisation of the hyper-parameters, Journal of Machine Learning Research, volume 8, pages 841-861, April 2007.

[23] G. C. Cawley and N. L. C. Talbot, Over-fitting in model selection and subsequent selection bias in performance evaluation, Journal of Machine Learning Research, vol. 11, pp. 2079-2107, July 2010

[24] Ruslan Salakhutdinov and Geoffrey Hinton. Deep Boltzman Machines.

[25] G. E. Hinton and R. R. Salakhutdinov. Reducing the dimensionality of Data with Neural Networks. Science, VOl 313, July 2006.

[26] CS231n Convolutional Neural Networks for visual Recognition

[27] Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks.

[28] Stéphane Mallat. Understanding Deep Convolution Networks. Philosophical transaction A, Jannuary 2016.

[29] Wei Wang and al.SINGA : Putting Deep Learning in the Hands of Multimedia Users

[30] Omry Y. and al.Multi-GPU Training of ConvNets Facebook AI group.

[31] Léon Bottou.Large-scale machine learning wih stochastic gradient descent

A
Code source

A.1 Logistic Regression

function [alltheta]= oneVSall(X, y, numlabels, lambda)

// X : input, Y : output , lambda : regularization parameter

$m = size(X, 1)$; // Get the input

$n = size(X, 2)$; // Get the output

$alltheta = zeros(numlabels, n + 1)$;

$X = [ones(m, 1)X]$; // The bias parameter

for $k = 1 : numlabels$

$initialtheta = zeros(n + 1, 1)$;

$options = optimset('GradObj',' on',' MaxIter', 500)$; // Gradient descent with 500 iterations

$[theta] = fmincg(@(t)(lrCostFunction(t, X, (y == k), lambda)), initialtheta, options)$; // fmincg to optimize the computing of cost function

$alltheta(k, :) = theta'$; // Get the derivative of theta

end

end

Figure 14: One-VS-All function Code

Function[J, grad]= CostFunction(theta, X, y, lambda)

$m = length(y)$;// Nnumber of training examples

$J = 0$;

$grad = zeros(size(theta))$;

$Stheta = sigmoid(X * theta)$;

// cost function

$J = (1/m) * sum(-y. * log(Stheta) - (1 - y). * log(1 - Stheta)) + (lambda/(2 * m)) * norm(theta([2 : end]))^2$;

$G = (lambda/m). * theta$;

$G(1) = 0$;

// Gradient

$grad = ((1/m). * X' * (Stheta - y)) + G$;

$grad = grad(:)$;

end

Figure 15: Cost function for logisitic regression Code

function [g]= sigmoid(z)

$g = 1.0./(1.0 + exp(-z));$

end

Figure 16: Sigmoid function

function [b]= predictOneVsAll(alltheta, X)

$m = size(X, 1);$

$numlabels = size(alltheta, 1);$

$p = zeros(size(X, 1), 1);$

$X = [ones(m, 1)X];$

$[x, p] = max(sigmoid(X * all_theta'), [], 2);$

b = p;

end

Figure 17: Predict One-VS-All Code

A.2 Multi layer Perceptron

$imageTrain = loadMNISTImages('train - images.idx3 - ubyte');$

$labelTrain = loadMNISTLabels('train - labels.idx1 - ubyte');$

$imageTest = loadMNISTImages('t10k - images.idx3 - ubyte');$

$labelTest = loadMNISTLabels('t10k - labels.idx1 - ubyte');$

$trainSize = size(imageTrain);$

$testSize = size(imageTest);$

$N = trainSize(2);$

$TOTALHU = 500;$

$TOTALIN = trainSize(1);$

$TOTALOUT = 10;$

$MAXITERATION = 100;$

$beta = 0.01;$ // Beta represent a scaling factor in sigmoid function

$alph = 0.1;$ //alpha represents the learning rate

$w1 = rand(TOTALIN + 1, TOTALHU);$ // Set the weights for the second layer

$w2 = rand(TOTALHU + 1, TOTALOUT);$ // Set the weights for the third layer

// Create variables to store weights

$x1 = zeros(TOTALIN + 1, 1);$

$x2 = zeros(TOTALHU + 1, 1);$

$x22 = zeros(TOTAHU, 1);$

$x3 = zeros(TOTALOUT, 1);$

$e1 = zeros(TOTALIN + 1, 1);$

$e2 = zeros(TOTALHU + 1, 1);$

$e3 = zeros(TOTALOUT, 1);$

$dw1 = zeros(TOTALIN + 1, TOTALHU);$

$dw2 = zeros(TOTALHU + 1, TOTALOUT);$

$A = eye(TOTALOUT);$

$success = zeros(MAXITERATION, 1);$

Figure 18: Multi Layer Perceptron Code Par1

// Start the learning from training set

for $t = 1 : MAXITERATION$

$perm = randperm(N)$;

for $c = 1 : N$

// feed forward propagation

$index = perm(c)$;

$x1 = [imageTrain(:, index); 1]$;

$x22 = sigmf(w1' * x1, [BETA, 0])$;

$x2 = [x22; 1]$;

$x3 = sigmf(w2' * x2, [BETA, 0])$;

// back propagation learning

$e3 = x3 - A(:, labelTrain(index) + 1)$;

$e2 = w2 * (e3. * x3. * (1 - x3))$;

$e2 = e2(1 : TOTALHU)$;

$e1 = w1 * e2$;

// Updating weights after back propagation learning

$w2 = w2 - ALPHA * x2 * (e3. * x3. * (1 - x3))'$;

$w1 = w1 - ALPHA * x1 * (e2. * x22. * (1 - x22))'$;

end

// Compute the accuracy and check the model

$success(t) = 0$;

for $i = 1 : N$

$x1 = [imageTrain(:, i); 1]$;

$x2 = [sigmf(w1' * x1, [beta, 0]); 1]$;

$x3 = sigmf(w2' * x2, [beta, 0])$;

$m = max(x3)$;

if $(m == labelTrain(i) + 1)$

$success(t) = success(t) + 1$;

Figure 19: Multi Layer Perceptron Code Part2

Here is the last part

```
end;end;end
accuracytrain = (success/trainSize(2)) * 100;testSuccess = 0;
for i = 1 : testSize(2) x1 = [imageTest(:, i); 1]; x2 = [sigmf(w1' * x1, [beta, 0]); 1];
x3 = sigmf(w2' * x2, [beta, 0]); [dummy, m] = max(x3);
if(m == labelTest(i) + 1)
testSuccess = testSuccess + 1;
end;end
accuracytest = (testSuccess/testSize(2)) * 100; fprintf('/ntestAccuracy : %f/n', accuracytest);
```

Figure 20: Multi Layer Perceptron Code Part3

A.3 Support Vector Machine

```
trainset = loadMNISTImages('train − images.idx3 − ubyte');
trainlabel = loadMNISTLabels('train − labels.idx1 − ubyte');
testset = loadMNISTImages('t10k − images.idx3 − ubyte');
testlabel = loadMNISTLabels('t10k − labels.idx1 − ubyte');
model = svmtrain(trainlabel, trainset,' − s0 − t1 − d2 − g1 − r1');
[predictedlabel, accuracy] = svmpredict(testlabel, testset, model);
```

Figure 21: SVM with polynomial Kernel Code

A.4 Convolutional Neural network

```
a.alpha = 1e-1; a.batchsize = 150; a.numepochs = 5;
a.imageDim = 28; a.imageChannel = 1; a.numClasses = 10;
a.lambda = 0.0001; a.momentum = .95;
a.mom = 0.5; a.momIncrease = 20;
images = loadMNISTImages('rain-images-idx3-ubyte');
images = reshape(images,a.imageDim,a.imageDim,1,[]);
labels = loadMNISTLabels('train-labels-idx1-ubyte');
labels(labels==0) = 10;
testImages = loadMNISTImages('t10k-images-idx3-ubyte');
testImages = reshape(testImages,a.imageDim,a.imageDim,1,[]);
testLabels = loadMNISTLabels('t10k-labels-idx1-ubyte');
testLabels(testLabels==0) = 10;
testImages = testImages(:,:,:,1:1000);
testLabels = testLabels(1:1000);
images = images(:,:,:,1:6000);
labels = labels(1:6000);
// call the library
cnn.layers =
struct('type', 'c', 'numFilters', 6, 'filterDim', 5) //convolution layer
struct('type', 'p', 'poolDim', 2) //pooling layer
struct('type', 'c', 'numFilters', 8, 'filterDim', 5) //convolution layer
struct('type', 'p', 'poolDim', 2); //pooling layer;
cnn = InitializeParameters(cnn,a);
cnnTrain(cnn,images,labels,testImages,testLabels);
```

Figure 22: Convolutional neural network main Code