Classification of Hand-Written Digits Using Scattering Convolutional Network - Project Proposal

Dongmian Zou
Email Address: zou@math.umd.edu
Department of Mathematics, University of Maryland, MD 20742

Advisor: Professor Radu Balan
Email Address: rvbalan@math.umd.edu
Department of Mathematics, University of Maryland, MD 20742

October 15, 2015

Abstract

In image classification it is advantageous to build classfiers based on feature extractors. It is demanded that the feature extractors unmask the identity of the image and remain stable with regard to small-scale deformations. A scattering convolutional network is such a feature extractor. In this project, we propose to use modern machine learning techniques to train the scattering convolutional network and test them on a standard database (MNIST) for the task of recognizing hand-written digits.

1 Background

Recently convolutional neural networks (ConvNets) have enjoyed conspicuous success in various pattern recognition tasks ([6], [7]). Although a clear understanding of why they perform so well is still absent, it is believed that it is the multi-layer structure that makes ConvNets outstanding ([3], [4]). As a feature extractor, a ConvNet is built to catch different features of an image in different layers. In [5], the authors apply wavelet theory to formulate a struture named Scattering Convolutional Network, which falls in the general categetory of ConvNets. The authors showed that the feature extractors they built are approximately invariant to translation and stable to small-scale deformation. The theory is extended to general semi-discrete frames in [9]. We are going to apply these theories to perform classification of hand-written digits.

1.1 Convolutional Networks

Convolutional networks (ConvNets) are artificial neural networks that "use convolution in place of general matrix multiplication in at least one place" ([1], Ch. 9). In one dimension, the convolution operation "$*$" is defined by

$$(x * h)(t) = \int x(s)h(t-s)ds \ .$$

In the above, despite the symmetric role of x and h, we usually call x the input, and h the filter. A digital image is a discretized two-dimensional object, whence we consider the convolution operator in two dimension with the discrete measure:

$$(X * H)(t_1, t_2) = \sum_{s_1} \sum_{s_2} X(s_1, s_2) H(t_1 - s_1, t_2 - s_2) \ .$$

In ConvNets, generally X is the input to the current layer and H is the filter which is commonly of much smaller size than X. The reason is both for computational efficiency and for sparse interactions between pixels ([1], Ch. 9). Since the size of the filter is small, it makes more sense to use the equivalent definition for the convolution, that is,

$$(X * H)(t_1, t_2) = \sum_{s_1} \sum_{s_2} X(t_1 - s_1, t_2 - s_2) H(s_1, s_2) \ . \tag{1}$$

Here t_1, t_2, s_1, s_2 are the pixel positions. Suppose $X \in \mathbb{R}^{D \times D}$ and $H \in \mathbb{R}^{d \times d}$ (we can replace $\mathbb{R}$ by $\mathbb{C}$ for some applications, but it does not affect our discussions here), in practice we would consider $D > d$. It is useful to think of X as a function $X : \mathbb{Z}_D \times \mathbb{Z}_D \to \mathbb{R}$. Then it makes sense when $t_1 - s_1$ or $t_2 - s_2$ is a negative number in (1).

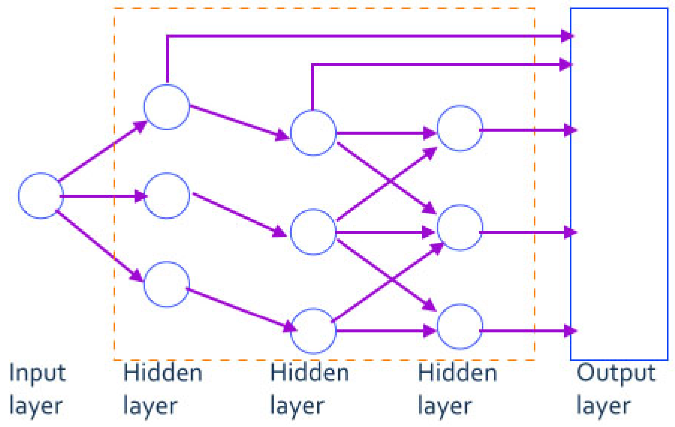

Figure 1: A typical ConvNet

Figure 1 illustrates a typical conlutional network. We see that between the input and the output it is natural to partition the neurons (operations) into several layers. In general, each layer of a ConvNet has three stages. The first stage is a convolution stage, where the input is convolved with filters; the second stage is a detector stage, where nonlinearity is applied to the output from the convolution stage; the last stage is a pooling stage, where local maximizing or averaging is applied to the output from the detector stage.

1.2 Scattering Convolutional Networks

Scattering Convolutional Networks are in the category of ConvNets with pre-defined filters ([5]). The filters are built by dilations of a wavelet. In the detector stage in each layer, the nonlinearity is chosen to be the absolute value function. In the pooling stage, a local averaging is done by convolution with a low-pass filter.

Let ψ be a wavelet in $L^2(\mathbb{R}^d)$. The dilation of ψ by λ, ψ_λ, is defined by

$$\psi_\lambda(t) = \lambda^d \psi(\lambda t) \ .$$

Consider a path $q = (\lambda_1, \lambda_2, \cdots, \lambda_m)$. The scattering propagator, $U[q]$, is defined by

$$U[q]x = |||x * \psi_{\lambda_1}| * \psi_{\lambda_2}| \cdots \psi_{\lambda_m}| \ \ ,$$

and the scattering transform, $S[q]$, is defined by

$$S[q]x = U[q]x * \phi_J \ ,$$

where $J > 0$ is some pre-determined scale. For an input signal x, in the m-th layer, a path $q = (\lambda_1, \lambda_2, \cdots, \lambda_m)$ generates an output given by $S[q]x$. Figure 2 showes the process.

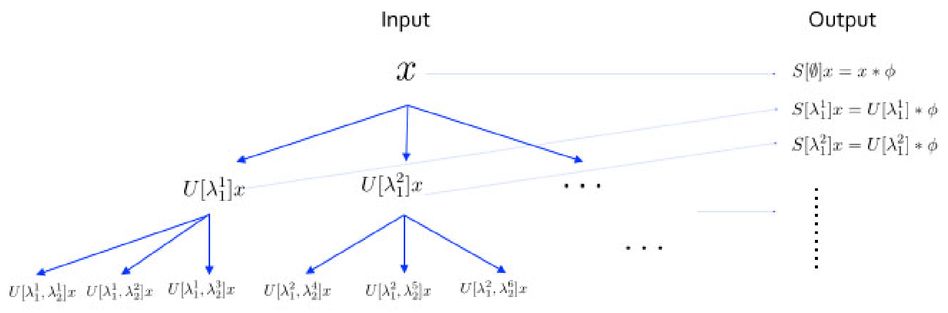
Input
Output
x
$S[\emptyset]x = x * \phi$
$S[\lambda_1^1]x = U[\lambda_1^1] * \phi$
$S[\lambda_1^2]x = U[\lambda_1^2] * \phi$
$U[\lambda_1^1]x$
$U[\lambda_1^2]x$
$U[\lambda_1^1, \lambda_2^1]x$
$U[\lambda_1^1, \lambda_2^2]x$
$U[\lambda_1^1, \lambda_2^3]x$
$U[\lambda_1^2, \lambda_2^4]x$
$U[\lambda_1^2, \lambda_2^5]x$
$U[\lambda_1^2, \lambda_2^6]x$

Figure 2: A typical scattering convolutional network

The Scattering Convolutional Networks are shown to be approximately tranlation invariant and stable to small-scale deformation. However, the stabily are guaranteed only if we choose a specifically designed wavelet to start with. In [9], the authors extend the theory to general semi-discrete frames. They use the same network structure but give a less strict condition on the filters.

1.3 Machine Learning Techniques

Our project is motivated by the above scattering network and its extension. Instead of fixing everything in the network beforehand, we want the computer to "tell" us what parameters to use for this structure. For instance, if we want to relate the input x and the output y by some function $y = f(x)$, then the process for determining the precise form of f is called the *training phase* ([2], Ch. 1). From the perspective of human, we "train" the system; while from the perspective of the machine, it "learns" from the given data set. For our project, we are given a set of training data, where each image has a "label" that tells us the de facto output.

The machine learning techniques we will use are not limited to but including the gradient descent optimization, the error backpropagation and the support vector machine.

1.3.1 Gradient Descent Optimization

In most machine learning tasks we would like to find the minimum of a *loss function* (also called a *cost function*). Suppose $l(\boldsymbol{\lambda})$ is the loss function with respect to the parameter $\boldsymbol{\lambda}$. The gradient descent is a iterative method for locating the minimum of l (see [1], Ch. 8; [2], Ch. 5). The updating step is given by

$$\boldsymbol{\lambda}^{(\nu+1)} = \boldsymbol{\lambda}^{(\nu)} - \eta \nabla l(\boldsymbol{\lambda}^{(\nu)}) \ . \tag{2}$$

where (ν) is the step number. At each step, $\boldsymbol{\lambda}$ travels towards the direction of the steepest descent. The scalar η is called the *learning rate* by the machine learning community. It is the same as the *stepsize* in standard optimization literature.

There are some alternative optimization methods, for example, conjugate gradients and quasi-Newton methods. They are in general more efficient than the gradient descent method. However, for training convolutional networks, we usually have a large amount of training data. Therefore, the loss function will be the sum of a large number of terms if we use deterministic methods. Taking this into consideration, deterministic methods are rarely used for network training. In practice, most training algorithms use stochastic approximation for the gradient. The conjugate gradients and quasi-Newton methods are not amenable to stochastic gradients. Nevertheless, it has been empirically demonstrated that the stochastic gradient descent method performs well for training ConvNets ([6], [7]). We will use this method for our project and we give the details in Section 2.

1.3.2 Error Backpropagation

The idea of *error backpropagation* is that the partial derivative of the loss function $l(\boldsymbol{\lambda})$ with respect to λ can be decomposed ([1], Ch. 6). It is based on the chain rule of taking derivatives.

Figure 3 epitomizes a simple neural network. The output y is connected to the input x through two intermediate outputs z_1 and z_2. Hence we have

$$\frac{\partial y}{\partial x} = \frac{\partial y}{\partial z_1}\frac{\partial z_1}{\partial x} + \frac{\partial y}{\partial z_2}\frac{\partial z_2}{\partial x} \ .$$

It is as if we had a network in the opposite direction for the partial derivatives. In general, we compute the partial derivatives between each connected

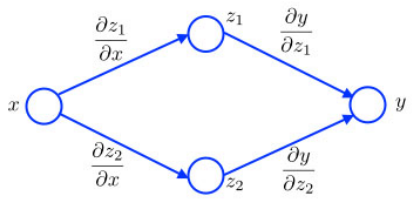
x
z_1
z_2
y
$\frac{\partial z_1}{\partial x}$
$\frac{\partial y}{\partial z_1}$
$\frac{\partial z_2}{\partial x}$
$\frac{\partial y}{\partial z_2}$

Figure 3: A simple illustration of backpropagation

neurons, and then propagate backward along the network to get the derivative of the loss function with respect to the parameters that we want to train.

1.3.3 Support vector machine

The support vector machine (SVM) is a decision machine designed for two-class classification problems ([2], Ch. 7). A typical trained SVM has two parameters: the weights w and bias b. Suppose the two classes are given by $\{\pm 1\}$, the given an input y, the sign of $\langle w, y\rangle + b$ determines whether y falls into the class $\{-1\}$ or $\{+1\}$.

We would like to train the SVM parameters $\{w, b\}$. The idea of the SVM is to maximize the margin between the decision boundaries. It turns out that we need to maximize $\|w\|^{-1}$, that is, to minimize $\|w\|$. However, at the same time, we want to softly penalize points that go to the wrong side. A standard way to do this is the following: suppose the inputs provided by the training data are $\{y_n\}_{n=1}^N$ with labels $\{a_n\}_{n=1}^N$ where each label $a_n \in \{\pm 1\}$, we want to solve

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N} l(y_n, a_n; w, b) \ ,$$

where C is a pre-determined parameter and l is the hinge-loss function defined by

$$l(y, a; w, b) = \max(0, 1 - a(b + \langle w, y\rangle)) \ ,$$

where $t \in \{\pm 1\}$ is the label corresponding to y.

2 Approach

Our task is the classification of hand-written digits. In this project, we will build a scattering convolutional network to extract the features of the images and an SVM to do the classification job. The input of the SVM is the output of the scattering network.

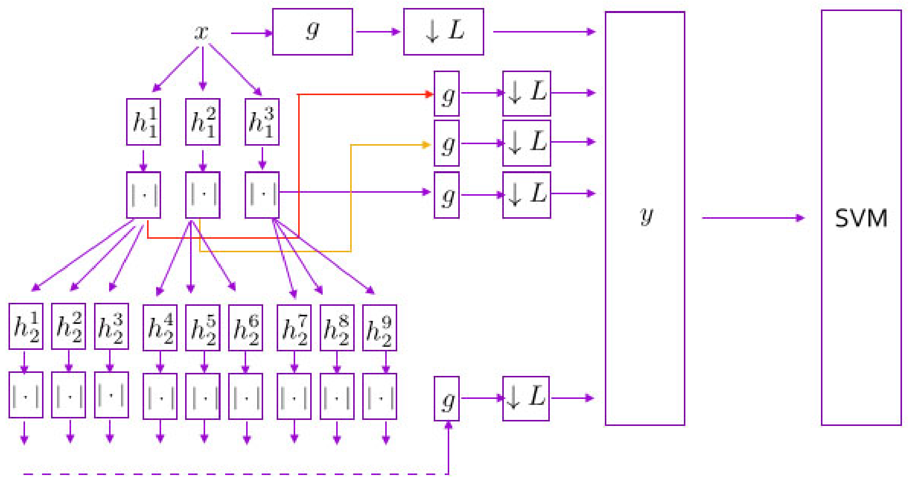

Figure 4: The structure of our scattering convolutional network

Figure 4 shows the detailed structure. The scattering network is a four-layer neuron network that consists of one input layer, two convolutional layers and one output layer. In each convolutional layer, h_k^j is the filter to be trained; "$|\cdot|$" is the operation of taking absolute value pointwise; g is a fixed low-pass filter that does local averaging. A downsampling is done before we send the output to the SVM (this guarantees that the output of the scattering network is appoximately of the same size as the input). We plan to choose g and L from several possible candidates after cross-validation.

As we have discussed in Section 1.3, once we have trained the network in Figure 2, we can use it for the classification task. We input the image x and let it propagate through the network. It first goes through the scattering network to generate y, and then the SVM determines which class x belongs to (by evaluating $\langle w, y\rangle + b$).

In our project, the two-dimensional filters h_k^j is parametrized as dilations of the tensor products of two one-dimensional wavelets. We use the same pre-defined wavelet ψ for both. That is,

$$h_k^j(t_1,t_2) = \psi_{\lambda_{k,1}^j} \otimes \psi_{\lambda_{k,2}^j}(t_1,t_2) = \lambda_{k,1}^j \lambda_{k,2}^j \psi(\lambda_{k,1}^j t_1)\psi(\lambda_{k,2}^j t_2) \ .$$

Let $\boldsymbol{\lambda}$ denote the vector composed of all the λ's in the h's. The training process deals with the unknown parameters $\boldsymbol{\lambda}; w, b$.

The optimization problem to solve for training our network comes from the training of the SVM, which we have discussed in Section 1.3. Note that the input y to the SVM now contains unknown parameters $\boldsymbol{\lambda}$. Therefore, the optimization problem for our project is as follows.

$$\min_{\boldsymbol{\lambda}; w, b} \quad \frac{1}{2}\|w\|^2 + C\sum_{n=1}^{N} l(y_n, a_n; w, b) \ , \tag{3}$$

where

$$l(y, a; w, b) = \max(0, 1 - a(b + \langle w, y\rangle)) \ ,$$

and y is the grouping of the following
(recall that "$*$" is in the sense of Equation (1))

$$\begin{aligned} y_0 &= x * g \ ; \\ y_1^j &= \left|x * h_1^j\right| * g \ , 1 \le j \le 3 \ ; \\ y_2^j &= \left|\left|x * h_1^{\lceil j/3\rceil}\right| * h_2^j\right| * g \ , 1 \le j \le 9 \ , \end{aligned}$$

where the two-dimensional filters h_k^j is parametrized as

$$h_k^j(t_1, t_2) = \lambda_{k,1}^j \lambda_{k,2}^j \psi(\lambda_{k,1}^j t_1)\psi(\lambda_{k,2}^j t_2) \ .$$

In the above, N is the total number of training data for deterministic methods, or the number of samples for stochastic methods. The input of the network is N pairs of (x_n, a_n) where x_n's are images of 28×28 pixels (according to the database) and a_n's are the corresponding labels indicating which class x is in. The unknown parameters are the λ's in the filters and the w and b in the SVM. The optimization problem with respect to $\{\boldsymbol{\lambda}, w, b\}$ is a non-convex problem and difficult to solve. Nevertheless, if we fix $\boldsymbol{\lambda}$, then training the SVM is a convex optimization problem. Inspired by this fact, we will follow a two-step procedure. Specifically, we first fix $\boldsymbol{\lambda}$ and train w and b (which is convex), and then fix w and b to train $\boldsymbol{\lambda}$ (which is easier than the original problem); and then iterate it until our stop criterion is met.

For the first step, we will use libSVM to train the SVM. LibSVM is a publicly available (https://www.csie.ntu.edu.tw/~cjlin/libsvm/) software for training SVM's. The released package includes the MATLAB interface.

The starting values for w and b will be randomly generated. We fix the regularization parameter C in (3) before calling libSVM. Each time we call the libSVM, we input the y_n's which are the output of the scattering network from the input training data x_n's.

Our crucial task is the second step, that is, training the filters. We have 12 filters and thus 24 parameters to train in the convolution layer. The starting value of $\boldsymbol{\lambda}$ is randomly gererated subject to the constraint that parameters in the same layer should be sufficiently apart. Note that w and b are now fixed, so the minimization problem (3) is turned into

$$\min_{\boldsymbol{\lambda}} \quad \sum_{n=1}^{N} l(\boldsymbol{\lambda}; x_n) \ ,$$

where we use

$$l(\boldsymbol{\lambda}; x_n) := l(y_n, a_n; w, b), \ \forall n$$

to emphasize the fact that l depends on the unknown parameters $\boldsymbol{\lambda}$ and the training data x_n's.

The updaing step follows Equation (2) while the learning rate η in (2) might be adjusted with steps. We have discussed the necessity of a stochastic gradient descent method. Therefore, at each step we sample N data from the training set and compute $\sum_{n=1}^{N} \nabla_{\boldsymbol{\lambda}} l(\boldsymbol{\lambda}; x_n)$ as the gradient.

We now compute the gradient $\nabla_{\boldsymbol{\lambda}} l(\boldsymbol{\lambda}; x)$ for a fixed x. Note that the loss function $l(y, t; w, b)$ is not a smooth function with respect to y. We use a smooth function in place. One possible choice is a C^1 function L defined by

$$L(y, a; w, b) = \begin{cases} 0.5 - a(b + \langle w, y \rangle) & , \text{if} \quad a(b + \langle w, y \rangle) \leq 0; \\ 0.5(1 - a(b + \langle w, y \rangle))^2 & , \text{if} \quad 0 < a(b + \langle w, y \rangle) \leq 1; \\ 0 & , \text{otherwise}. \end{cases}$$

Also, we need to use some function F in place of $|\cdot|$ when we relate y to $\boldsymbol{\lambda}$. One possible choice is a C^∞ function $F : \mathbb{R} \to \mathbb{R}$ defined by $F(t) = (|t|^2 + \epsilon^2)^{1/2}$ for some small ϵ. We use F (resp. F') for the operation of applying F (resp. F') pointwise as well (the same treatment as for $|\cdot|$). Moreover, we use $\Psi(\lambda)$ defined by $\Psi(\lambda)(t) := \lambda\psi(\lambda t)$ to emphasize the variable λ. Note that

$$\frac{\partial L}{\partial \lambda_{k,i}^j} = \left\langle \nabla_{y_k^j} L, \frac{\partial y_k^j}{\partial \lambda_{k,i}^j} \right\rangle \ .$$

Now we give expressions for $\partial y_k^j / \partial \lambda_{k,i}^j$. We use $\odot$ to denote pointwise multiplication of two vectors or matrices of the same size (which is an abuse of notation, but we do not plan to use $\odot$ elsewhere). Since i takes value in $\{1,2\}$, we use i' to denote the complement of i in $\{1,2\}$. We have

$$\frac{\partial y_1^j}{\partial \lambda_{1,i}^j} = \Big[F'\Big(x * (\Psi(\lambda_{1,i}^j) \otimes \Psi(\lambda_{1,i'}^j))\Big) \odot \Big(x * (\Psi'(\lambda_{1,i}^j) \otimes \Psi(\lambda_{1,i'}^j))\Big)\Big] * g \ ;$$

$$\begin{aligned}\frac{\partial y_2^{3j-\iota}}{\partial \lambda_{1,i}^j} = \Big\{&F'\Big(F\Big(x * (\Psi(\lambda_{1,i}^j) \otimes \Psi(\lambda_{1,i'}^j))\Big) * \Big(\Psi(\lambda_{2,i}^{3j-\iota}) \otimes \Psi(\lambda_{2,i'}^{3j-\iota})\Big)\Big) \odot \\ &\Big[\Big[F'\Big(x * (\Psi(\lambda_{1,i}^j) \otimes \Psi(\lambda_{1,i'}^j))\Big) \odot \Big(x * (\Psi'(\lambda_{1,i}^j) \otimes \Psi(\lambda_{1,i'}^j))\Big)\Big] \\ &* \Big(\Psi(\lambda_{2,i}^{3j-\iota}) \otimes \Psi(\lambda_{2,i'}^{3j-\iota})\Big)\Big]\Big\} * g \ , \quad \text{for } \iota = 1,2,3;\end{aligned}$$

$$\begin{aligned}\frac{\partial y_2^j}{\partial \lambda_{2,i}^j} = \Big[&F'\Big(F\Big(x * (\Psi(\lambda_{1,i}^{\lceil j/3 \rceil}) \otimes \Psi(\lambda_{1,i'}^{\lceil j/3 \rceil}))\Big) * \Big(\Psi(\lambda_{2,i}^j) \otimes \Psi(\lambda_{2,i'}^j)\Big)\Big) \odot \\ &\Big(F\Big(\Big(x * (\Psi(\lambda_{1,i}^{\lceil j/3 \rceil}) \otimes \Psi(\lambda_{1,i'}^{\lceil j/3 \rceil}))\Big)\Big) * \Big(\Psi'(\lambda_{2,i}^j) \otimes \Psi(\lambda_{2,i'}^j)\Big)\Big)\Big] * g \ .\end{aligned}$$

The above expression is simply given by the chain rule. We can compute $\partial L/\partial \lambda_{k,i}^j$ by backpropagation as described in Section 1.3. A sketched illustration is given in Figure 5. Together with the above expression, the process should be clear.

The following diagram summarizes our algorithm.

Algorithm 1: The algorithm for network training

```
Start with learning rate $\eta$, regularization parameter $C$ ;
randomly generate $\boldsymbol{\lambda}, w, b$;
while stop criterion not met do
    sample $N$ examples $\{x_1, x_2, \cdots, x_N\}$ from the training set;
    propagate forward to get $\{y_1, y_2, \cdots, y_N\}$;
    call libSVM with input $\{y_1, y_2, \cdots, y_N\}$ and $C$;
    update $w, b \leftarrow$ output of libSVM;
    set $\boldsymbol{r} = 0$;
    for $n = 1$ to $N$ do
        compute $\nabla_{\boldsymbol{\lambda}} l(\boldsymbol{\lambda}; x_n)$;
        $\boldsymbol{r} \leftarrow \boldsymbol{r} + \nabla_{\boldsymbol{\lambda}} l(\boldsymbol{\lambda}; x_n)$;
    update $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} - \eta \boldsymbol{r}$ ;
    adapt $\eta$ accordingly.
```

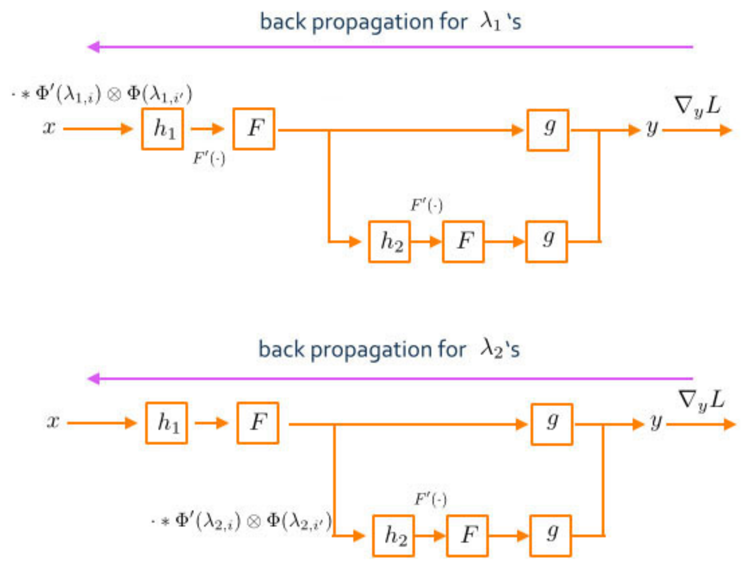
back propagation for λ_1's
$\cdot * \Phi'(\lambda_{1,i}) \otimes \Phi(\lambda_{1,i'})$
x
h_1
F
$F'(\cdot)$
g
y
$\nabla_y L$
h_2
back propagation for λ_2's
$\cdot * \Phi'(\lambda_{2,i}) \otimes \Phi(\lambda_{2,i'})$

Figure 5: Sketched illustration of the backpropagation process ("·" denotes the input from the last node)

3 Implementation

We will implement our algorithm on the personal laptop with

- CPU: 2GHz Intel Core i7
- Memory: 8 GB 1600 MHz DDR3
- OS: OS X El Capitan Version 10.11
- Software: MATLAB R2015b

4 Database

The database we use for both training and testing is the standard MNIST database. MNIST is a publicly available database (http://yann.lecun.com/exdb/mnist/) of hand-written digits. The images are already preprocessed and formatted: they are of 28×28 pixels; each pixel has an value ranging from 0 to 255. There are 60,000 images for training and 10,000 for testing. Examples of images are given in Figure 6.

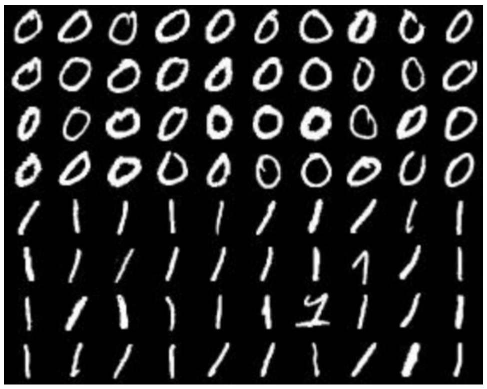

Figure 6: Examples of training images of 0's and 1's in the MNIST Database, retrieved from http://www.cs.nyu.edu/˜roweis/data.html

5 Validation

We will the MatConvNet Toolbox to validate our training model. MatConvNet is publicly available (http://www.vlfeat.org/matconvnet/). We will use its built-in functions to train the network in Figure 2 and compare with our model trained with the approach discussed in Section 2.

6 Testing

We will use the testing images in the MNIST database for testing our trained network. A reasonable measure is the percentage of errors. We will run libSVM independently (i.e. using SVM for classification without a feature extractor) and compare the results. We are first going to do it for the 2-class task. Time permitting, we are going to compare the multi-class results.

7 Project Schedule and Milestones

- September - October 2015: Define the project. Investigate in existing literature. Design the algorithm.
- November 2015: Write codes for training the convolutional filters.
- December 2015: Write codes for classfication for 2 classes (multi-class if time permits).
- February 2016: Complete validation.

- March 2016: Complete testing.
- April 2016: Wrap up the project.

8 Delivarables

At the end of the project, we should be able to deliver:

- the datasets
- the toolboxes
- the MATLAB codes
- the trained network
- the results
- the proposal, reports, presentation slides, etc.

References

[1] Y. Bengio, I. J. Goodfellow and A. Courville, *Deep Learning*, Book in preparation for MIT press, 2015.

[2] C. M. Bishop, *Pattern Recognition and Machine Learning*, New York: Springer, 2006.

[3] J. Bruna et al., *A Theoretical Argument for Complex-Valued Convolutional Networks*, submitted, 2015.

[4] J. Bruna and S. Mallat, *Invariant Scattering Convolution Networks*, Pattern Analysis and Machine Intelligence, IEEE Transactions 35(8) (2013), 1872–1886.

[5] S. Mallat, *Group Invariant Scattering*, Comm. Pure Appl. Math., 65 (2012): 1331-1398.

[6] A. Krizhevsky, I. Sutskever, G. E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*, Advances in Neural Information Processing Systems 2 (2012): 1097-1105.

[7] C. Szegedy et al, *Going Deeper with Convolutions*, Open Access Version available at
http://www.cv-foundation.org/openaccess/content_cvpr_2015/
papers/Szegedy_Going_Deeper_With_2015_CVPR_paper.pdf,
retrieved October 9, 2015.

[8] A. Vedaldi and K. Lenc, *MatConvNet - Convolutional Neural Networks for MATLAB*, Proc. of the ACM Int. Conf. on Multimedia, 2015.

[9] T. Wiatowski and H. Bölcskei, *Deep Convolutional Neural Networks Based on Semi-Discrete Frames*, Proc. of IEEE International Symposium on Information Theory (ISIT), Hong Kong, China, pp. 1212–1216, June 2015.