Gradient-Based Learning Apllied to Document recognition - An overview

Jan Ruman

21.4.2021

Abstract

This work attempts to summarize the Gradient-Based Learning Apllied to Document recognition work by LeCun et. al. The main emphasis will be put on describing models predating the work as well as models introduced by the work.

1 Overview

The main goal of the work is to show the superiority of gradient-based methods as opposed to handcrafted heuristics and algorithms in the field of document recognition. There have been several works showing great result of gradient-based learning applied to handwritten character recognition predating this work and the work continues in this line by introducing a new model called Graph Transformer Network, that enables a global (ie end-to-end) learning. To show that this model exceeds the performance of handcrafted systems in practice, the authors also include a case study concerned with automated reading of bank checks.

2 Introduction

In the introduction the authors provide an overview of the limitations of handcrafted heuristics and algorithms for the task of character recognition, namely the infeasibility of handcrafting the necessary rules that would capture the immense variability and complexity of real world data, as well as the neccessity to go over such daunting process for every real world application, as the rules rarely apply outside of one single use case. Using these limitations, as well as previous accomplishments of gradient-based methods, they argue for a more automated process using GTNs.

Rest of the introduction is devoted to machine learning concepts. Aside basic ideas like learning from data and gradient-based learning they introduce one of the central components of this work - Globally Trainable Systems. These are based on the principle of building a globally differentiable system out of individual differentiable components (that is differentiable with respect to their trainable parameters and input).

The basic principle of Globally Trainable Systems is chain rule. Given $X_n = F_n(W_n, X_{n-1})$ representing an output of given module as a function of the component's weights W_n and output of previous component X_{n-1} (or input if it is the first component in a cascade). Then, given a vector of computed loss E^p we can easily compute the partial derivatives with respect to both weights and inputs of each component:

$$\frac{\partial E^p}{\partial W_n} = \frac{\partial F}{\partial W}(W_n, X_{n-1})\frac{\partial E^p}{\partial X_n}$$

$$\frac{\partial E^p}{\partial X_{n-1}} = \frac{\partial F}{\partial X}(W_n, X_{n-1})\frac{\partial E^p}{\partial X_n}$$

3 Convolutional neural networks

In the next section, authors review the concept of convolutional neural networks and a one of their implementations, LeNet-5. We are going to cover both.

3.1 General CNN

Even though regular neural networks could be used for individual character recognition, they have several shortcomings:

- no sense of topological structure of the input
- translational variance with respect to input
- non-locality
- size

All of these problems solve convolutional neural networks. A convolutional neural network typically consists of 3 types of layers - convolutional layer, subsampling layer (sometimes called pooling) and fully connected layer. First layers of the network are typically pairs of convolutional layer and subsampling layer; last layers are typically fully connected (and equivalent to layers in regular neural networks).

An input to the CNN is typically a 2D image, even though it's architecture enables an input of arbitrary dimensionality. A convolutional layer consists of a number of small matrices (also called kernels), typically 3x3 or 5x5, whose values are trainable parameters. Convolutional layer applies every matrix to the input such that the kernel is matched with every possible position in the input image and for each of those positions the values of kernel and the values on corresponding coordinates in the input are multiplied and summed together; a learnable bias is also added. Using this procedure we obtain a set of feature

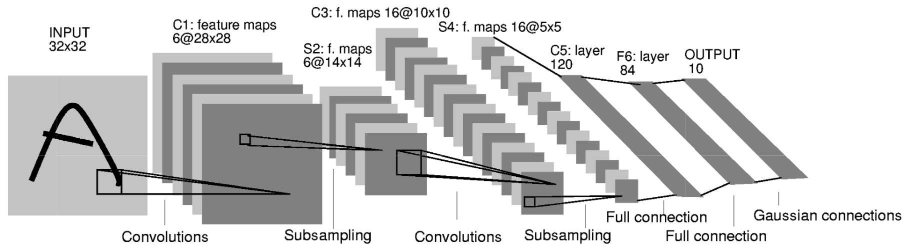

Figure 1: Architecture of LeNet-5.

maps, one for each kernel, that itself is 2D (we ensure that by assigning a position for each output value corresponding to the location of original input).

A subsampling layer is very similar to the convolutional layer, except the kernel values are all ones, the output is mutliplied by trainable coefficient, trainable bias is added and the output is passed through a sigmoidal function. The kernels are also applied in a non-overlapping manner starting from top left. For each input feature map, a convolutional network typically has a subsampling kernel applied just to the corresponding input feature map. This means that the number of feature maps doesn't change by subsampling, even though the feature maps themselves get smaller.

In the beginning, we said how convolutional layer is applied to a single 2D input; however, it's output are several 2D feature maps, so we might wonder how the next convolutional layer would deal with that. That depends on particular architecture, but one way to do it would be to apply every kernel to every feature map, resulting in $\#feature\ maps \times \#number\ kernels$ output feature maps; a sum over the feature maps would also be possible, equalizing the number of output feature maps and number of kernels. LeNet-5 implements a different regimen, which we will observe below.

Lastly, how do fully connected layers fit into all this? A careful reader might've noticed that both convolutional and subsampling layers shrink the sizes of feature maps. CNNs are typically designed in a way that results in many 1x1 feature maps toward the end of the network. A set of such feature maps can be viewed as a vector, which is perfectly fine as a fully connected layer input. This also means that outputs can be handled the same way they are handled in the case of regular neural networks (eg using softmax for classification).

3.2 LeNet-5

LeNet-5 is a particular implementation of the convolutional neural network, with several interesting design choices. Figure 1 depicts the structure of LeNet-5. We are going to progress from the input to the output layer, describing individual components of the network. We are going to refer to the individual layers according to their labels as shown on image (eg C1 for first convolutional layer).

	0	1	2	3	4	5
0	X	X	X			
1		X	X	X		
2			X	X	X	
3				X	X	X
4	X				X	X
5	X	X				X
6	X	X	X	X		
7		X	X	X	X	
8			X	X	X	X
9	X			X	X	X
10	X	X			X	X
11	X	X	X			X
12	X	X		X	X	
13		X	X		X	X
14	X		X	X		X
15	X	X	X	X	X	X

Table 1: Mapping of kernels to feature maps.

The input size is assumed to be 32x32. Layer C1 consists of six 5x5 kernels, resulting in six feature maps of size 28x28. A subsampling layer with 2x2 kernels follows, outputing six 14x14 feature maps.

Here comes first LeNet-5 specific design choice: kernels of C3 aren't applied in each-to-each fashion; rather, each of the 16 kernels is applied to a subset of feature maps (as indicated in Table 1) and the outputs corresponding to a given subset are summed up. This way we arrive at sixteen 10x10 feature maps.

The subsampling layer S4 consists of kernels of size 2x2, shrinking the size of feature maps to 10x10.

The last convolutional layer C5 consists of 120 kernels, each applied to every input feature map and the outputs are summed up, resulting in 120 1x1 feature maps - which is exactly what we want, as the following layers are fully connected.

F6 is just a regular fully connected layer with 84 units.

Very interesting design choice awaits at the end of the network. The output layer is fully connected with F6, however this time the values of units don't correspond to dot products between input vector and weight matrix; instead, the weights are fixed values and the output value of a unit y_i is an RBF, that is

$$y_i = \sum_j (x_j - w_{ij})^2$$

This value corresponds to a Euclidean distance between x and $w_{i,*}$. The weights for the last layer are carefully selected. For each output unit, it's weight vector consist of pixel values corresponding to a 7x12 bitmap of the number that is assigned to this particular unit. This is also the reason the previous layer has 84 units.

It is worth noting that this model isn't trained with the Maximum Likelihood Estimation criterion, that is, not with pure MLE. The MLE criterion is in this case augmented by a "competetive" term, which forces the network to output the correct answer with as big activation as possible. The term is

$$\log(e^{-j} + \sum_i e^{y_i(Z^P, W)})$$

The j is a positive constant and $y_i(Z^P, W)$ is the i-th output unit value, as computed from input vector Z^p and weight matrix W.

3.3 Results

The authors tested LeNet-5 on the MNIST dataset with success (we won't dwelve into the particular values here). The authors have also noted that adding augmented examples increased the accuracy of the trained model. LeNet-5 scored either better or on par with other models used (most notable was SVD).

4 Multi-module systems, GTNs

The authors propose a novel architecture called Graph Transformer Network. As it's name indicates, it is collection of connected Graph Transformers (the network itself forms a directed acyclic graph), each of which takes a graph and outputs a graph. These components aren't necessarily completely differentiable (eg switch), but they are to the extent that a derivative exists almost everywhere with respect to weights and input.

The network itself works primarily with graphs as inputs (and some components output graphs). The motivation for that is simple - working with vectors of fixed size wouldn't be possible with variable length input. Instead of fixed size vectors, graphs describing sequences of vectors are used.

The authors propose a simple GTN architecture for classification of handwritten sequences of characters. Given an image of handwritten character sequence a heuristic over-segmentation is used to produce a graph G_{seg}. This graph is fed into a recognition transformer T_{rec} that for each possible segment from G_{seg} assigns penalties for each possible character and creates an interpration graph G_{int} where every single arc is replaced by set of arcs corresponding to characters and their respective penalties. Next a Viterbi transformer is used to find a path in the graph with smallest penalty - which corresponds to a sequence of characters. Figure 2 illustrates this architecture.

In this model a CNN can be used in the step where segments are classified as characters.

The training procedure for this model is non-trivial. First step is to add a so called Path Selector transformer between Recognition and Viterbi transformer. The Path Selector will be used during training to select only paths corresponding to correct assignment and the length of the shortest such path is set to be the loss. This on itself isn't sufficient - to minimize the loss all the model needs to do

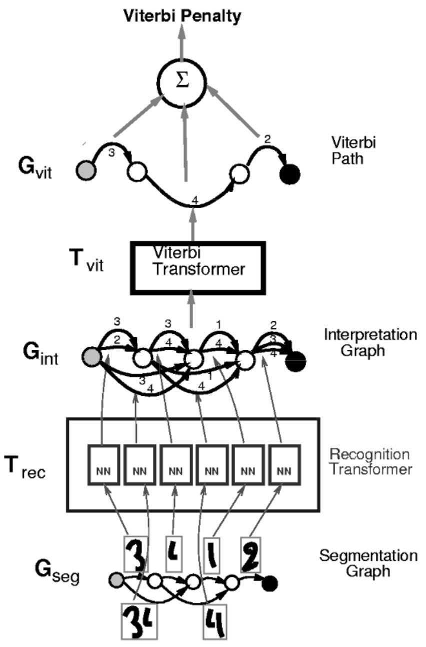

Figure 2: Simple GTN architecture.

is to assign small penalty to all arcs. That on itself isn't an optimum solution, however, as authors note, this solution constitutes a saddle point which is easy to get into but hard to get out of.

We would like the loss to also contain information about the penalties of other sequences, because the model ought to maximize their penalties and it would also make the loss a more reasonable metric. We can subtract the penalty of the best solution from the penalty of correct solution - that would be a much better indicator of the quality of the trained model. An adjusted model can be seen of Figure 3.

Authors also propose several more refinements to the training procedure; however, we won't go over those here.

5 Space displacement NN

An alternative to the model proposed above is the so called Space Displacement Neural Network. Instead of heuristic over-segmentation it "slides" a window accross the input image (assumed to contain a sequence of characters) and let's a CNN compute output for every position of the window (note the positions might overlap). The neural network thus outputs a distribution of penalties over possible character sequences.

Such a graph would contain many "gibberish" characters as well as same character being recognized several times in consecutive runs of CNN (thanks to it's robustness). To create a more reasonable graph authors propose a composition of the SDNN output graph and a grammar transducer, which helps to identify which paths in the graph are more likely.

6 Usage in practice

The authors applied their models to an on-line handwriting recognition system and a check reading system.

For the on-line handwriting recognition both the simple GTN architecture introduces earlier and SDNN were used. Even though the heuristic over-segmentation outperformed the SDNN, the authors make an interesting observation - that character error rates dropped in both cases upon introducing global training schedule.

The solution for check reading system consists of several parts: a field location transformer that heuristically extracts rectangular zones that may contain the check amount, segmentation transformer that segments the fields using heuristic methods, recognition transformer consisting of LeNet-5 for character recognition, composition transformer selecting valid character sequences and Viterbi transformer that selects the path with smallest penalty. Even though this system scored on par with other systems on a selected test set, the integrators of these systems chose this system based on another independent test.

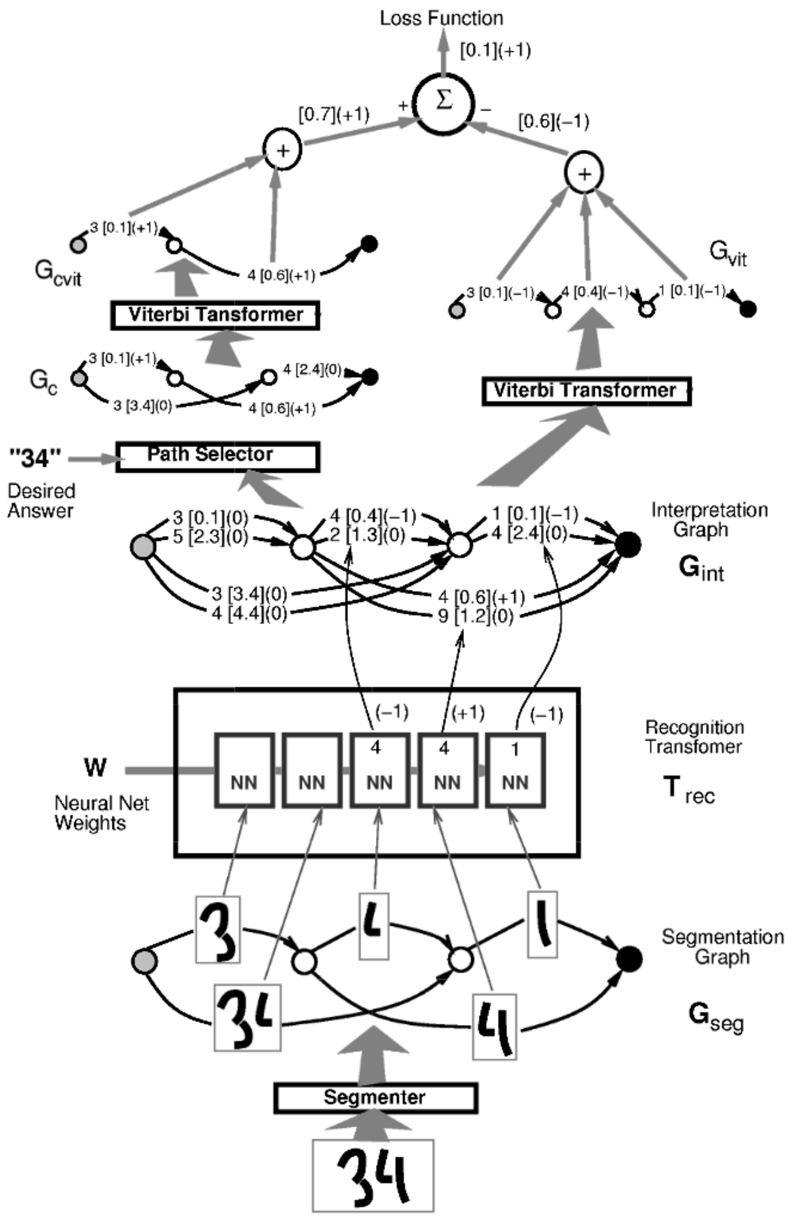

Figure 3: GTN architecture adjusted for training.

7 Conclusions

The authors conclude the work by reiterating the superiority of gradient-based methods, placing emphasis on global training as opposed to the training of individual components of given system. One of the core concepts of these methods, the authors acknowledge, is that they are generic, that is they require minimum of prior task specific knowledge.

The authors also recall the basic problems in pattern recognition systems, namely: finding a low dimensional set of salient features (feature extraction), the inherent connection between segmentation and recognition, data labeling, ambiguities in segmentation. GTNs try to tackle each of these issues in some way, with a relative success.