Final Project Report

Shengdong Zhang[*]
Department of Computing Science
Simon Fraser University
Burnaby, BC Canada V5A 1S6
sza75@sfu.ca

Abstract

My project consists of 3 major components: visualization of higher order features, generating digits, and globally optimizable auto-encoders. I visualized higher order features in 2 different ways —— optimization and approximation. Based on the visualized features, I implemented a digit generator. Lastly, I developed a new type of auto-encoders. This type of AE's has the property of global optimizability.

1 Visualization of Higher Order Features

1.1 The Optimization Method

When we want to investigate what feature can fire a neuron in a hidden layer, namely, what feature is able to maximize the output of the neuron, it is natural to randomly generate a point in the input space and search for such feature by gradient ascent. This idea was also proposed by [Erhan, 2009]. However, due to non-convexity, the found feature by this optimization method does not guarantee to be the global maximum. In addition, it requires the explicit formula of gradient. Assuming all activation functions are sigmoid, denoted by $Sig(\cdot)$, below are the formulas. Since visualizing the first order features is trivial, I only give the formulas for the neurons in the second layer and above.

To maximize a neuron in the second layer, the gradient formula is

$$\tfrac{d}{dx}h_{2j}(x|\theta) = W_1^T diag(Sig(\vec{z}_1) \bullet (1 - Sig(\vec{z}_1)))Sig(z_2)(1 - Sig(z_2))\vec{w}_j^{(2)}$$

To maximize a neuron in the third layer, the gradient formula is

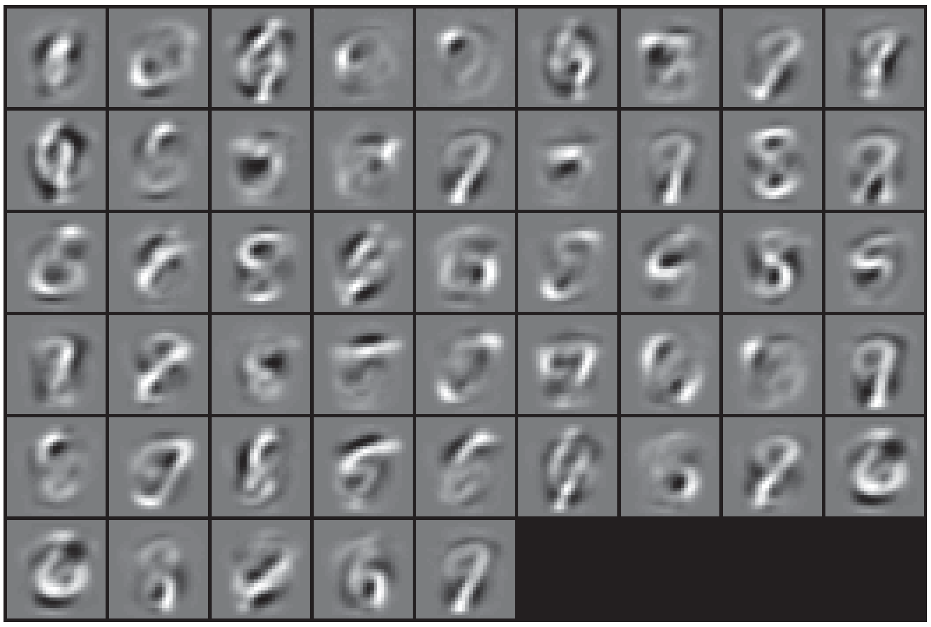
$$\tfrac{d}{dx}h_{3j}(x|\theta) = W_1^T diag(Sig(\vec{z}_1) \bullet (1 - Sig(\vec{z}_1)))W_2^T diag(Sig(\vec{z}_2) \bullet (1 - Sig(\vec{z}_2)))Sig(\vec{z}_3)(1 - Sig(\vec{z}_3))\vec{w}_j^{(3)}$$

To maximize a neuron in the N-th layer, the gradient formula is

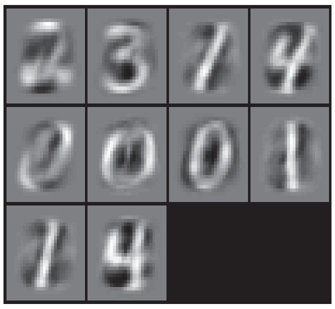
$$\tfrac{d}{dx}h_{Nj}(x|\theta) = \prod_{i=1}^{N-1} W_i^T diag(Sig(\vec{z}_i) \bullet (1 - Sig(\vec{z}_i)))Sig(z_N)(1 - Sig(z_N))\vec{w}^{(N)}$$

Below are the higher order features learned from the MNIST digit data set.

[*]I very appreciate Dr. Schulte's valuable suggestions and helps.



Second Order Features



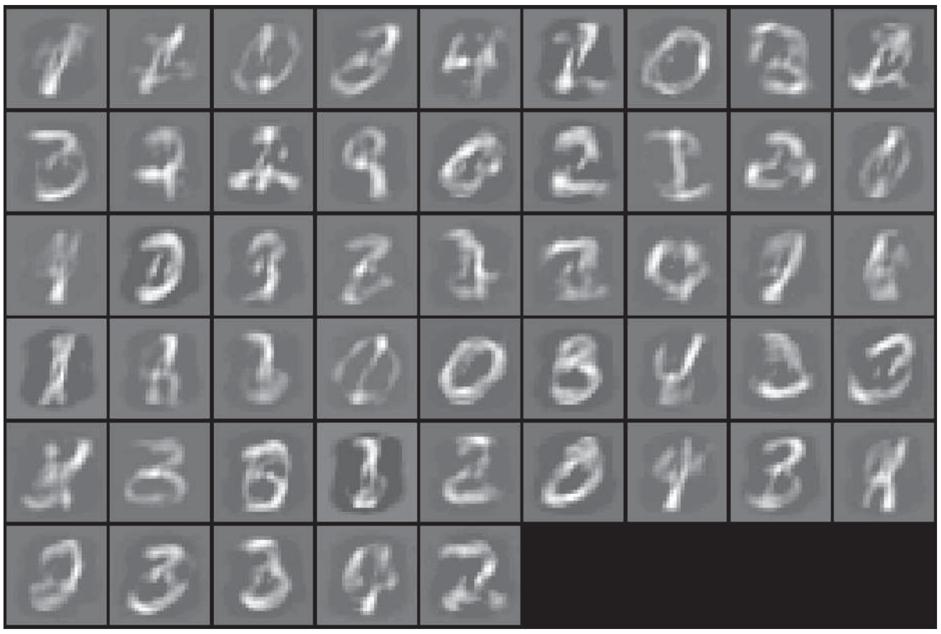
Third Order Features

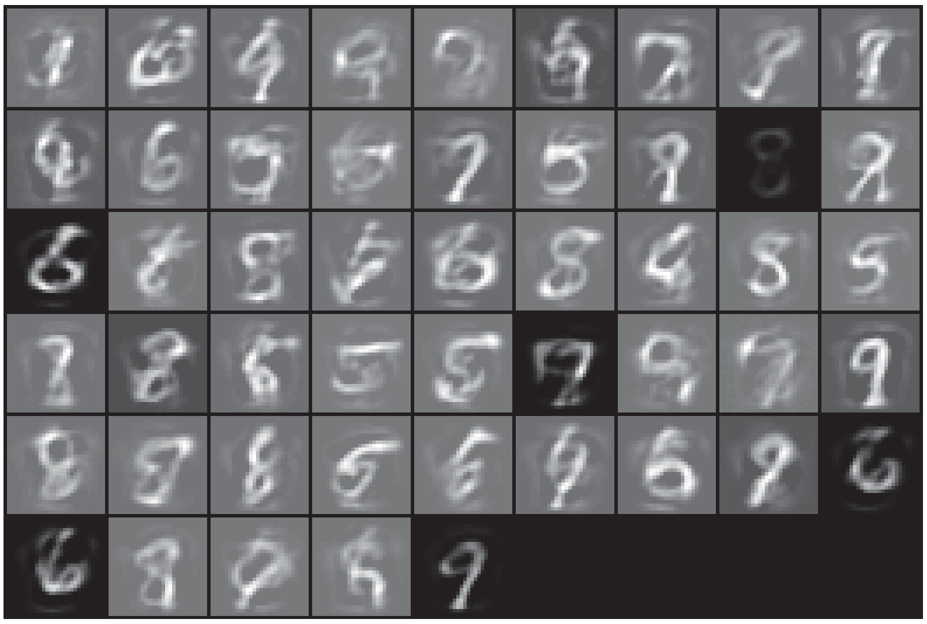
1.2 The Approximation Method

This method does not require computing gradient. However, it takes longer time than the optimization method for searching for solutions to multiple constrained least square problems. Below is the example used in my presentation. Assume we are maximizing a hidden neuron and it outputs 0.9 as being fired. Below is the algorithm.

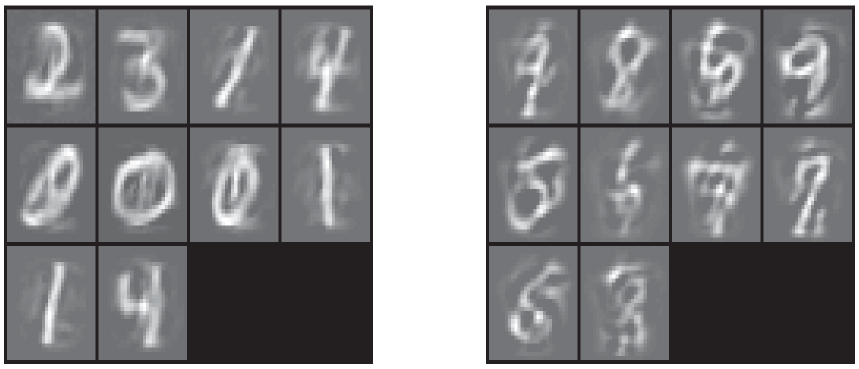
- $z_3 = h^{-1}(0.9)$
- $\vec{a}_2 = arg\,min||\vec{w}^{(3)T}\vec{a} + b^{(3)} - z_3||_2^2 \; s.t. \; 0 < a_i < 1$
- $\vec{z}_2 = h^{-1}(\vec{a}_2)$
- $\vec{a}_1 = arg\,min||\vec{W}^{(2)}\vec{a} + \vec{b}^{(2)} - \vec{z}_2||_2^2 \; s.t. \; 0 < a_i < 1$
- $\vec{z}_1 = h^{-1}(\vec{a}_1)$
- $\vec{x}_{opt} = arg\,min||\vec{W}^{(1)}\vec{a} + \vec{b}^{(1)} - \vec{z}_1||_2^2 \; s.t. \; 0 < x_i < 1$

There are many available packages to find the solution to constrained least square problem. Below are the approximated higher order features learned from the MNIST digit data set.





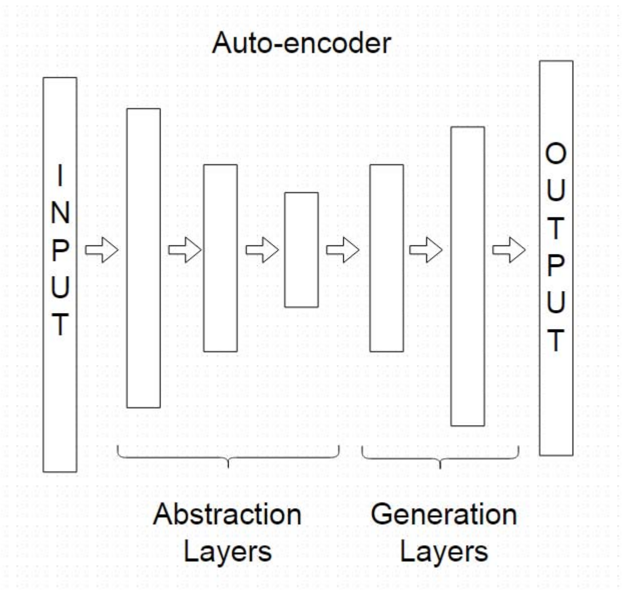
Approximate Second Order Features

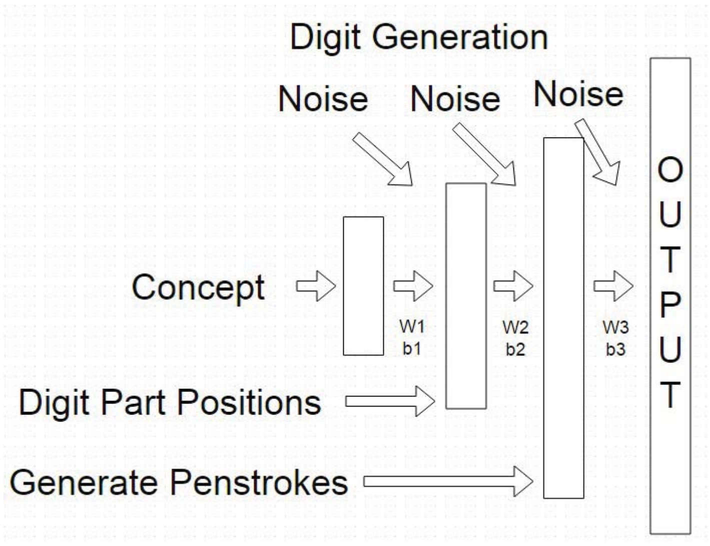


Approximate Third Order Features

Compared the approximate features with the exact ones, they look poor and it seems approximation method does not do a good job on the feature visualization task. However, my point is we can actually use the output of the approximation method as the input to the optimization method. Simply generating a random vector and then doing gradient ascend with it has no guarantee what feature this vector will converge to. Therefore, we should start gradient ascend search with the vector containing prior information. By combining these two methods together, I think we could get better results.

2 Digit Generation





Because an AE tries to learn an identity function from data, once it is trained, it should be able to re-generate data to some extent. To re-create the data generating process, we must use those generation layers, which are always disposed after training the AE. Unlike RBM or DBN, since everything of an AE is deterministic, to make the generating process look like the digit generator made by Dr. Hinton, when the signal propagates from the concept layer to the output layer, we need to purposely add noise to the input of each layer except the concept layer. That is,

$$Output = Sig(W3\, Sig(W2\, Sig(W1\, \vec{c} + \vec{b}1 + \vec{\epsilon}) + \vec{b}2 + \vec{\epsilon}) + \vec{b}3 + \vec{\epsilon})$$

where Wi and bi are weight matrices and bias vectors corresponding to different layers. $\vec{\epsilon}$ represents noise. The added noise level can't be too high, otherwise all signals will be corrupted.