NeuroTextures:

Image Completion and Storage Using Neural Networks

M. Erol Aran

merol@cs.bilkent.edu.tr

Bilkent University

Abstract: Recovery of corrupted images, large image databases and texture synthesis are three diverse topics that are studied in both computer graphics and in computer vision. In this paper we demonstrate to teach a neural network simply a texture to handle the problems stated. In particular, we propose a novel approach, the NeuroTextures, to model the texture at hand. NeuroTextures not only model the image but it is effective to complete the parts of an image when it is corrupted. Different network architectures are offered for modeling. We modeled just a simple intensity texture with a NeuroTexture, but its extensions to medical data, photovolumetric data and image databases are straightforward.

Keywords: Textures, neural networks, learning.

1 Introduction

Application of machine learning methods to computer graphics is relatively a new topic whereas there have been lots of research in the computer vision field in the last decade. Images are stored as to represent pixel values in computers. Although there is no problem in storing and processing small images, manipulation and storage of large images, medical data and large image databases are still challenging problems in the literature. Beside storage and processing problems there is always the problem of recovery of images that are corrupted.

This paper proposes neural network approaches to store images and complete corrupted images within NeuroTextures, whose name was inspired by Demetri Terzopoulos' NeuroAnimators [1]. In this work we trained NeuroTextures that learn only a small (ie;32x32 8bpp, 64x64 8bpp) corrupted texture, but the work presented here may be especially to corrupted medical data perhaps deserving the name NeuroTexels etc.

1.1 Previous Work

We do not see much work taken in computer graphics field related with network architectures. Network models especially found place in computer graphics literature by Demetri Terzopoulos et. Al. Before his work, some researchers used simple architectures in character animation. Ridsdale [2] uses a connectionist model of skill memory for skill acquisition. Van de Panne and Fiume [3] uses the sensor-actuator networks that are recurrent networks of units taking sensory information as input and producing actuator controls as output. Sims designed a network architecture to construct simple “brains” that control evolved creatures.

Except from those explained above, Terzopoulos et. al exploited neural networks to produce controlled, physically realistic animation satisfying user-specified constraints at a fraction of the computational cost of conventional numerical simulation. Their work tried to replace the numeric solution of complex animation system with a fast neural network emulator.

Inspired by these results, we thought using neural networks for volume, medical or simply texture data. Medical data especially comes in image slices that are sampled from a patient’s body’s some region. Each slice stores some data related to the region of interest in terms of X-Ray absorption, ultra sound interference etc. and this information is usually stored as 8 or 16 bit integers to represent the relative intensity of the measured activity. Modeling through neural networks give us the opportunity to reduce the memory cost as much as completing the missing information in these images.

1.2 NeuroTextures

Teaching a neural net a texture can be done through supervised learning where each of the training sample’s class label is known. Then another important parameters that effect the network performance are the number of training samples, type of the network, number of input and output layer units, number of hidden layers and hidden layer units etc. Feature selection is another important topic that we consider.

In designing NeuroTextures we try different network architectures to get the best performance in terms of low training time and correct classification of pixel intensities. Before discussing the different types of architectures it is necessary to introduce some field related concepts.

2 Artificial Neural Networks (ANNs)

An Artificial Neural Network (ANN) is an information processing unit that is inspired by the nervous systems. ANNs learn by example. Learning in biological systems involves adjustments to the synaptic connections that exist between the neurons. The same approach is valid for the ANNs.

The basic processing unit of an ANN is called a neuron. Mathematically, the neuron behaves as an activation or mapping function f(.) producing an output y = f(*net*), where *net* is the cumulative input stimuli to the neuron and f is typically a nonlinear function of *net*. For example, *net* is often taken as the weighted sum of the inputs

$$net = x_1 w_1 + x_2 w_2 + x_3 w_3 = \sum_i x_i w_i$$

and f is typically a monotonic nondecreasing function of net. An artificial neuron model is shown below in Figure 1.

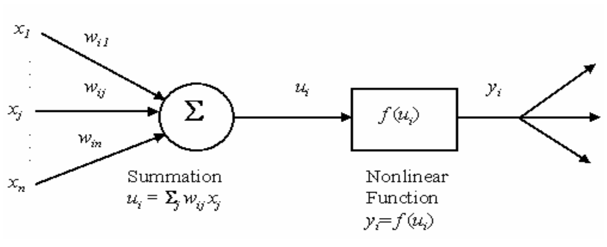


Figure 1 – Artificial Neuron

In our experiments we used the sigmoid function as the activation function. For sigmoid units**,** the output varies continuously but not linearly as the input changes. Sigmoid units bear a greater resemblance to real neurons than do linear or threshold units, but all three must be considered rough approximations. Some activation functions used in ANNs are in Figure 2.

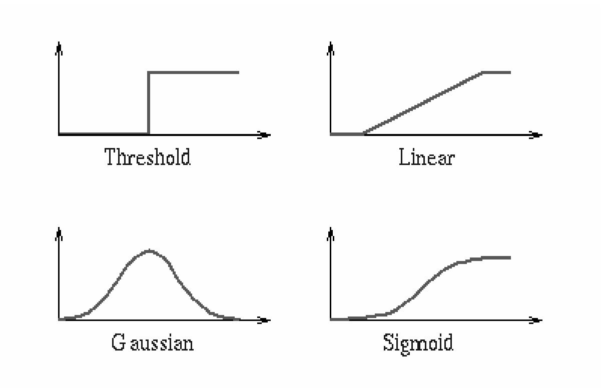


Figure 2 – Activation Functions

A neural network is a set of interconnected neurons. The connections between the units and the layers characterize the network. In our experiments we mainly used feedforward neural network.

2.1 NeuroTextures with Multilayer Feedforward Neural Networks

The first type of the network architecture that we used for designing a NeuroTexture is the multilayer feedforward neural network (MLFF) with backpropagation (BP) learning. A simple perceptron may also be used for a trial but since our textures represent arbitrary nonlinear regions, it does not worth trying such a simple model. Therefore we start with an MLFF. A general MLFF is shown in Figure 3.

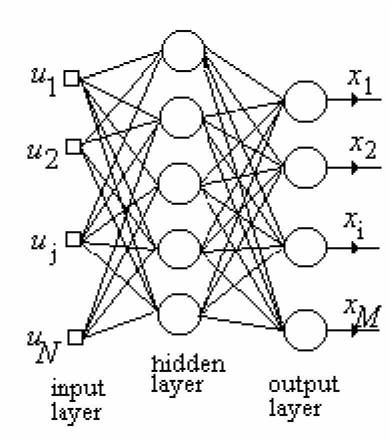


Figure 3 – A General Multilayer Feedforward Network

The network shown in Figure 3 is a feedforward, fully-connected hierarchical network consisting of an input layer, one hidden layer and an output layer. The internal layers are called ‘hidden’ because they only receive internal inputs (inputs from other processing units) and produce internal outputs (outputs to other processing units).

A unit in the hidden layer computes a function of the input signal and the weights, and passes its output forward to all of the units in the next successive layer. The process is repeated until the final computation is produced by the output units when there are more than one hidden layer.

To train the network there have been so many approaches such as the perceptron learning algorithm, Widrow-Hoff LMS or delta learning rule and backpropagation algorithm. Neither the perceptron algorithm nor the Widrow-Hoff delta rule cannot be applied to networks with hidden layers. These methods fail tospecify how to adjust the hidden layer weights. This is known as the ‘credit assignment’ problem since these nethods are unable to give credit or assign blame to hidden layer weights for errors that occur in the output layer. The backpropagation learning method, a generalization of the delta rule, does specify how to adjust the weights in hidden layers. It therefore permits the construction of feedforward, multilayer networks which can learn to compute much more complex mappings.

2.1.1 Input Layer for The NeuroTexture

To be able to determine the number of input units, it is necessary to decide the features that is representative for the problem domain. We approach the problem in a local and spatial way and choose the pixel coordinates as the features. Of course a more global approach is necessary for more complicated subjects such as texture synthesis or image segmentation, but for now we just focus on the learning of the texture and local completion. We found it to be sufficient for this purposes. As a result, our NeuroTexture has 2 input units in the input layer one of which is the x coordinate and other one is the y coordinate of the pixel.

2.1.2 Input and Output Layers

After selecting the features for a given texture image and inputs for the network, it is necessary to model the output values. For the sake of simplicity, we teach the network 8 bit intensity images only for now. Extension to 24 bit color images is obvious but requires more memory space and training time. In an 8 bit intensity image, the value of a pixel is between 0 and 255 meaning it can have 256 different values. This leads us to design the network with 256 output units in the output layer.

2.1.3 Training Examples and Sampling Process

Our training examples for the NeuroTexture is the pixel coordinates and corresponding intensity values. It is important here that if the network is

given all the pixels in the image, then it will try to overfit the data which will require a lot of neurons in the hidden layers. On the other hand, supplying too few data will prevent the network to learn the overall structure of the texture. Therefore, we give the network only a representative part of the image and this is controlled by horizontal and vertical sampling rates. The training data is formed by sampling the image in both horizontal and in vertical axes. For fixed network parameters, the best sampling frequency appeared to be 4 in our experiments. Frequencies above 4 result in very distorted images and the network cannot get the details well. Frequencies below 4 result in overfitting, requires a lot of neurons and long training times.
When the sampling frequency is selected as 4, it means that we construct the image from its 1/16 sized samples, meaning if we have 1/16 of an image, it is possible to recover it with NeuroTextures as long as the data is representative.

2.1.4 Hidden Layers

When designing the hidden layers, we first prefer the simpler approaches such as one hidden layer and networks having no more than 500 neurons in the hidden layer. We realized that for images larger than 64x64 pixels, it is more efficient to partition the image into parts and designing a NeuroTexel for each part. Small, simpler networks are more successfull in learning the images. For a 32x32 image, one hidden layer with 100 neurons with a sampling frequency of 4 in both horizontal and vertical axes achieved the best in our experiments. When the image size is increased to 64x64, it is necessary to partition the image and train each part separately or to add one more hidden layer to compensate for the reflection of the complexity of the image.

2.2 Backpropagation Learning

The backpropagation learning method was discovered by several researchers for different reasons. Werbos [4] was perhaps the earliest to propose the method. Even so, Rumelhart is known for his many contributions on the algorithm and putting it into work. The backpropagation learning method can be applied to any multilayer network that uses differentiable activation functions and supervised training. It is an optimization procedure based on gradient descent that adjusts weights to reduce the system error or cost function. During the learning phase, input patterns are presented to the network in some sequence. Each training pattern is propagated forward layer by layer until an output pattern is computed. The computed output is then compared to a desired or target output and an error value is determined. The errors are used as inputs to feedback connections from which adjustments are made to the synaptic weights layer by layer in a backward direction.
Using backpropagation, the hidden layer weights are adjusted using the errors from the subsequent layer. Thus, the errors computed at the output layer are used to adjust the weights between the

last hidden layer and the output layer. Likewise, an error value computed from the last hidden layer outputs is used to adjust the weights in the next to the last hidden layer and so on until the weight connections to the first hidden layer are adjusted. In this way, errors are propagated backwards layer by layer with corrections being made to the corresponding layer weights in an iterative manner. The process is repeated a number of times for each pattern in the training set until the total output error converges to a minimum or until some limit is reached in the number of training iterations completed.

2.2.1 Network Parameters of The NeuroTexture

2.2.1.1 Learning Rate Coefficient

The learning rate coefficient detemines the size of the weight adjustments made at each iteration and hence influences the rate of convergence. This coefficient is important since poor choice can result in failure to converge. Although it is known that it is better to have a varying learning rate coefficient throughout the learning process, for the sake of simplicity we kept it to be constant. We found it to be 0.00001 for the best convergence.

2.2.1.2 Momentum Constant

The rate of convergence also may be improved with introduction of some inertia or momentum to the gradient expression. This can be accomplished by adding a fraction of the previous weight change to the current weight change. The addition of such a term can help smooth out the descent path by preventing extreme changes in the gradient due to local anomalies. It can act as an averaging effect which smooths the trajectory of the gradient as it moves downhill. The value of the momentum constant should be positive and less than 1. In fact it is better to vary it during the learning phase but in our problems we found it to be nonproblematic for a constant momentum constant. In our problem we found the momentum constant to be 0.9 for the best convergence.

2.2.1.3 Activation Function

The most frequently used activation function in the neural network community is perhaps the sigmoid function. This function has some very good properties especially when the computational properties are concerned. Although it may drastically decrease the performance when used in the incorrect problem, this was not the case for ours. We designed all of our layers with a sigmoid activation function.

2.2 NeuroTextures with Boltzmann Machines and Simulated Annealing

Although the results with multilayer feedforward neural network with backpropagation are not bad, we think that we may get better convergence rates with less neurons using a different architecture. Our another approach to design the NeuroTexel is Boltzmann Machines.

Multilayer feedforward networks are good at generalizations and learning but when it comes to image remembering associative memory approaches are more successfull. We did not try associative memory directly because it is too simple to handle our problem. Hopfield networks which are similar to the Boltzmann machine is another alternative but we did not prefer it since it does not have hidden layers which we suppose the network should have for the diverse nonlinearity of textures.

2.2.1 Boltzmann Machines

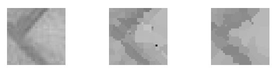
The Boltzmann machine is different from a feedforward neural network in the sense that it is a stochastic network. The states which the network assumes are governed by the Boltzmann distribution, an exponentional form of probability distribution which is used to model the states of a physical system at thermal equilibrium.

Feedforward networks with backpropagation had the risk of being trapped in a local minimum. Boltzmann machines tackle this problem. A method calles as simulated annealing is applied to the network during operation and learning. This process permits the network to escape from local minima and converge to a global equilibrium.

Design of the machine and its results will be given after the review process.

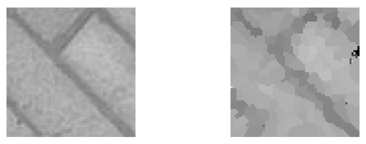
3 Results

We have experimented with 32x32 and 64x64 textures. The original textures and the learned textures are shown in Figure 4 and Figure 5.



(a) (b) (c)

Figure 4 – Original texture in (a). 2 hidden layers with 100 neurons total with a sampling frewuency of 3 (b). 1 hidden layer with 100 neurons total with a sampling frewuency of 4 (c)



(a) (b)

Figure 4 – Original texture in (a). 2 hidden layers with 2000 neurons total with a sampling frewuency of 4 (b)

4 Summary &Future Work

We presented a neural network approach to the completion, storage and synthesis of textures. This work may find application areas such as scientific visualization, image databases and perhaps compression.

As a future work, Boltzmann machine implementation is proposed. Also experiments with larger datasets and especially with medical data may be done. Performance analysis of the system may also be done more seriously.

References:

[1] NeuroAnimator: FastNeural Network Emulation and Control of Physics-Based Models,SIGGRAPH98,Radek Grzeszczuk,Demetri Terzopoulos, Geoffrey Hinton,

[2] G. Ridsdale. Connectionist modeling of skill dynamics. *Journal of Visualization and Computer Animation*, 1(2):66–72, 1990.

[3] Michiel van de Panne and Eugene Fiume. Sensor-actuator networks. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 335–342, August 1993.

[4] Werbos, Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences, 1974, Harvard University, doctoral dissertation