ECE 285 – MLIP – Project B Style Transfer

Bolin He*
Department of Electrical and Computer Engineering
University of California, San Diego
San Diego, CA 92093
Github Name: Bervvv
b2he@ucsd.edu

Kunpeng Liu
Department of Electrical and Computer Engineering
University of California, San Diego
San Diego, CA 92093
Github Name: Eazymemory
kul003@ucsd.edu

Bowen Zhao
Department of Electrical and Computer Engineering
University of California, San Diego
San Diego, CA 92093
Github Name: Soolizo
b2zhao@eng.ucsd.edu

Team name: Learning Machine

Github Link: https://github.com/Soolizo/ECE285_Style_Transfer

Abstract

In this project, we are aiming to develop neural network models to realize picture style transformation. Eventually with well-trained CNN Transfer model and Cycle-GAN model, we managed to transfer the artistic style of a photo while keeping its primary content unchanged. The first model based on *A Neural Algorithm of Artistic Style*(Gatys. et al, 2015), which needs several minutes to get the result from one style pictures and one content pictures. And the second model based on *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Network* which needs to train a Cycle-GAN for several hours on DSMLP with GPU, and based on the network after train, we can get a real-time style transfer(one style needs one network).

*Authors are ordered alphabetically by their last name

1 Part I: Neural Transfer

1.1 Introduction

The concept of style transfer is introduced in *A Neural Algorithm of Artistic Style*[2](Gatys. et al, 2015), then he also described more detail about how to use CNN in *Image Style Transfer Using Convolutional Neural Networks* (Gatys. et al, 2016). The purpose of the paper is to create artistic images using deep neural network. How to separate and recombine the contents and style of arbitrary images and how to define "style" are most challenging problem. Because of the lack of concrete semantic representation for images, traditional methods to extract style information is to extract texture features manually, which is so inefficient. A Neural Algorithm of Artistic Style allows us to solve these problems by using convolutional neural network which is already trained in the field of target recognition and Gatys used VGG-Network. In the first part of this article, we will reproduce the part of Gatys' experiments.

1.2 Methods

The 19 layer VGG-Network we used has 16 convolutional and 5 pooling layers and we won't use any fully connected layers for arbitrary size of images. And Gays also pointed out that by replacing the max-pooling operation with average pooling, the results were represented better.

1.2.1 Content representation

Giving that $\vec{x}$ and $\vec{p}$ are the original image and the generated image, P^l and F^l are their respective feature representation in layer l. Content loss was defined as the squared-error loss between the two feature representations

$$\mathcal{L}_{content}(\vec{p}, \vec{x}, l) = \frac{1}{2}\sum_{ij}(F_{ij}^l - P_{ij}^l)^2.$$

And the respective derivative of this loss in layer l equals

$$\frac{\partial \mathcal{L}_{content}}{\partial F_{ij}^l} = \begin{cases} (F^l - P^l)_{ij} & if \quad F_{ij}^l > 0 \\ 0 & if \quad F_{ij}^l < 0 \end{cases}.$$

Then by using standard error back-propagation we can compute the gradient.

1.2.2 Style representation

Gram matrix was chose to compute the feature correlations between different feature maps i and j in layer l:

$$G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l.$$

To generate a texture that matches the style of a given image, Gays used gradient descent of a white noise image to match the style of the original image. Given that $\vec{a}$ and $\vec{x}$ are the original image and the generated image, A and G are their respective gram matrix, so their mean-squared error equals

$$E_l = \frac{1}{4N_l^2 M_l^2}\sum_{i,j}(G_{ij}^l - A_{ij}^l).$$

And the total loss is

$$\mathcal{L}_{style}(\vec{a}, \vec{x}) = \sum_{l=0}^{L} w_l E_l.$$

Where w_l are weighing factors of the contribution of each layer Finally we need to minimize the sum of content loss and style loss which are weighted by α and β. Finally we need to minimize the sum of content loss and style loss which are weighted by alpha and beta.

$$\mathcal{L}(\vec{p}, \vec{a}, \vec{x}) = \alpha\mathcal{L}_{content}(\vec{p}, \vec{x}) + \beta\mathcal{L}_{style}(\vec{a}, \vec{x})$$

[2]https://arxiv.org/pdf/1508.06576.pdf

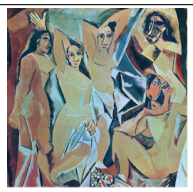
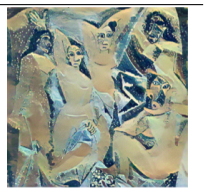
1.3 Experiment

In this article, we use Pytorch to reproduce the result of Gatys. As described above, we define content loss and style loss, and then insert these loss to pre-trained VGG network after the convolution layer they are detecting by create new sequential module. As Gatys suggested, we use L-BFGS algorithm to run gradient descent and create a Pytorch L-BFGS optimizer `get_input_param_optimier` and pass our image to it as the tensor to optimize. We choose *The Great Wave off* (Kanagawa, Hokusai 1829-1832) as style image and an arbitrary image as content image which is also used as input image (we can also choose a white noise image as input image as Gatys did). Then we set the ratio of content loss and style loss (α/β) to different values and set the epoch to 100 and print each loss.

1.4 Result

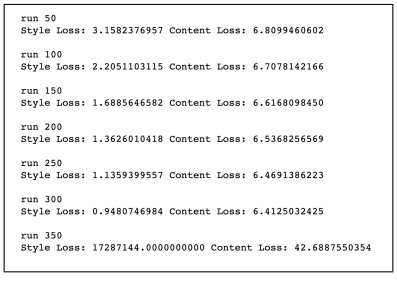
The results of our experiments as follow:

Table 1: the effect of different value of α/β on the generated images

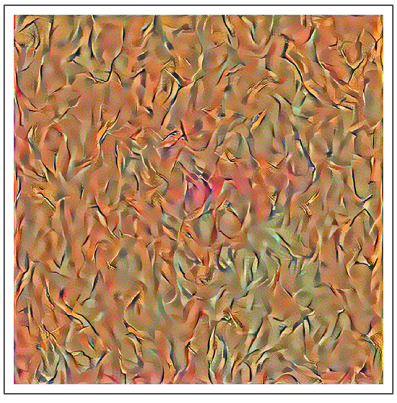
α/β	1	10^3	10^6	10^9
α*style loss	0.0004678459	0.1597140133	0.3083346188	13.678116798
β*content loss	0.0000000033	0.1480611712	1.6951342821	3.1885426044
generated image				

1.5 Disscusion

We reproduce part of Gatys' experiments successfully. As Table 1 shows, the style of generated images approaches the desired style of *The Great Wave off* with α/β increasing. In our initial experiments, We set the epoch to 500 with all the (α/β) ratios. Unfortunately, this is too slow to get the result (this may caused by the server of DSMLP) and when the epochs gets too large, this will cause many problems, such that to some styles, the style loss first decrease and after some epochs, it grows rapidly in the large ratio of (α/β). To solve these problems, we just decrease the epochs to 100 and this give us the good outputs.



(a) Rapid growth of loss



(b) One of the failure cases.

Figure 1: Failure case

2 Part II: Cycle-GAN

2.1 Introduction

In this part, we implemented a real-time Style Transfer using Cycle-GANs introduced in paper *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Network*[3]. Through profound training of our Cycle-GAN models, we are able to transfer the style of an input picture both based on a single style-source picture and a domain of style-souce pictures from an aritist.

The idea of Cycle-GAN is built on the idea of Generative Adversarial Network (GAN). In our task of style transferring, we will need two generator modules and two discriminator modules for style and content tranformation respectively. In addition to the steps of traditional GAN, Cycle-GAN also constrain our optimization task by applying two generator sequentially on the input and see if the output is the same as the input.

The more detailed implementation for the generator, discriminator, and the Cycle-GAN will be illustrated in the following sections.

2.2 Cycle-GAN Network

The main difference of Cycle-GAN and traditional GAN is the training process and the way we calculate the loss of each models in our network. Thus, we can implement a Generator class and a Discriminator class for Cycle-GAN as follows:

2.2.1 Generator

We adapted the DnCNN model with depth $\mathcal{D}$ defaulted to 6, and channel size $\mathcal{C}$ defaulted to 64 from Assignment 4 for our Generator network. More specifically, a Generator with depth $\mathcal{D}$ contains the following network layers in this order:

$1\times$ $\text{Conv}_{3*3}^{3\to\mathcal{C}}$ + ReLU
$\mathcal{D}\times$ $\text{Conv}_{3*3}^{\mathcal{C}\to\mathcal{C}}$ + BN + ReLU
$1\times$ $\text{Conv}_{3*3}^{\mathcal{C}\to 3}$

The forward method is implemented to apply the networks above in that order to the input tensor and output the resulting tensor.The criterion method is inherited from class DnCNN, but we never call this function in the training/testing process.

In this task, we created two Generator objects:

- `gen2s`: A generator transform the style of a picture to the style of style_ref
- `gen2c`: A generator transform the content of a picture to the content of input content

2.2.2 Discriminator

The Discriminator will be used to distinguish the "fake" image generated by a Generator. In order to achieve this goal, we define the Discriminator to have the following network layers in this order:

- $\text{Conv}_{3*3}^{3\to 16}$ + InstanceNorm(16) + LeakyReLU(0.2)
- $\text{Conv}_{3*3}^{16\to 64}$ + InstanceNorm(64) + LeakyReLU(0.2)
- $\text{Conv}_{3*3}^{64\to 256}$ + InstanceNorm(256) + LeakyReLU(0.2)
- $\text{Conv}_{3*3}^{256\to 256}$ + InstanceNorm(256) + LeakyReLU(0.2)
- $\text{Conv}_{3*3}^{256\to 1}$

The forward method is implemented to apply the networks above in that order to the input tensor, reshape the output tensor to shape (batch_size, image_w*image_h), and output the reshaped tensor. We also implemented the criterion method for the sake of implementing abstract class, but we never call this function in the training/testing process.

[3]https://arxiv.org/pdf/1703.10593.pdf

In this task, we created two Generator objects:

- `dis_s`: A discriminator distinguishing if a tensor is truly style_ref
- `dis_c`: A discriminator distinguishing if a tensor is truly input content

2.3 Training Method

2.3.1 Training Algorithm

The main algorithm behind the training of Cycle-GAN is an MiniMax adversary game between generators and discriminators. During the training process, we train our generator networks so that they will get better and better on generating the corresponding result tensor and fool the corresponding discriminator. In the mean time, we also train our discriminators so that it will get better and better on telling fake generated tensors from real inputted tensors.

This idea is implemented with calculating corresponding losses for both generator and discriminator, which will be explained more in detail in the section below. More generally, if we take a look at the full objective for Cycle-GAN, we will find that the training process is an MiniMax optimization where our objective is:

$$\begin{aligned}\mathcal{L}(gen2s, gen2c, dis\_s, dis\_c) = &\mathcal{L}_{GAN}(gen2s, dis\_s, content_{ref}, style_{ref}) \\ &+ \mathcal{L}_{GAN}(gen2c, dis\_c, content_{ref}, style_{ref}) \\ &+ \mathcal{L}_{cyc}(gen2s, gen2c, dis\_s, dis\_c, content_{ref}, style_{ref})\end{aligned}$$

and we want to solve for best style generator $gen2s^*$ such that

$$gen2s^* = \arg\min_{\substack{gen2s \\ gen2c}} \max_{\substack{dis\_s \\ dis\_c}} \mathcal{L}(gen2s, gen2c, dis\_s, dis\_c)$$

2.3.2 Trainer

We implemented class `CGANTrainer` to update the optimizer based on loss and train each model. To initialize a `CGANTrainer` object, we need to only pass in $\mathcal{D}$, will be used to specify the depth for Generator. The `CGANTrainer` object will store all input generator models and discriminator models as its class attribute. Thereafter, we defined three Adam optimizer: one for each discriminator, and one for the two generators together, with learning rate initialized to 1×10^{-3} and betas initialzed to (0.5, 0.999).

In addition, a `CGANTrainer` object has methods `train`, `update_lr`, `forward`, `train_generator`, and `train_discriminator`. In an experiment (class `CGANexp`), users only need to call `train` and `update_lr` to train the generators and discriminators.

2.3.3 Training Loss

Generator Loss Since we have one optimizer for the two generators together, we will calculate one total loss for the two generators together each time we call `train_generator`. The loss is broken into two parts: $\mathcal{L}_{cheat}$ and $\mathcal{L}_{Cycle}$.

$\mathcal{L}_{cheat}$ represent a generator's ability to make corresponding discriminator unable to tell the generated tensor from the real input tensor. This loss contains two parts: cheating dis_s and cheating dis_c. The loss of cheating dis_s is calculated as the MSE-Loss between $dis\_s\,(gen2s(content_{ref}))$ and a tensor filled with 1's. Similar calculation apply to calculating the loss of cheating dis_c. This loss will be minimized when the generator outputted the desired output such that the discriminator is unable to distinguish. Thus, we have the formula below:

$$\mathcal{L}_{cheat} = \mathbf{E}\left[(dis\_s(gen2s(content_{ref})) - 1)^2\right] + \mathbf{E}\left[(dis\_c(gen2c(style_{ref})) - 1)^2\right]$$

L_{Cycle} represent the ability of the generator to recover origin tensor if we apply the two generators sequentially. That is to say, under good generator, we expect $gen2c(gen2s(content_{ref}))$ to be close to $content_{ref}$ and $gen2s(gen2c(style_{ref}))$ to be close to $style_{ref}$. Thus, we have the formula below:

$$\begin{aligned}L_{Cycle} = &||gen2c(gen2s(content_{ref})) - content_{ref}||_1 \\ &+ ||gen2s(gen2c(style_{ref})) - style_{ref}||_1\end{aligned}$$

Thus, the total loss for generators' optimizer is $\mathcal{L}_{cheat} + \mathcal{L}_{Cycle}$.

Discriminator Loss Since we have one optimizer for the each discriminator, we need to calculate the loss for each discriminator separately and update the corresponding optimizer accordingly each time we call `train_discriminator`. We will use the loss of dis_s as an example, and the loss of dis_c is calculated in similar way. The loss for dis_s is broken into two parts: $\mathcal{L}_{identity}$ and $\mathcal{L}_{dis}$.

$\mathcal{L}_{identity}$ represent whether the discriminator knows the $style_{ref}$ inputed to the network is real. We simply call dis_s on $style_{ref}$ and calculate the MSE-Loss between the FC tensor outputted by dis_s and a tensor filled with 1's. Thus, we have the formula below:

$$\mathcal{L}_{identity} = \mathbf{E}\left[(dis\_s(style_{ref}) - 1)^2\right]$$

L_{dis} represent the ability of the discriminator to tell the "fake" tensor outputted by the generator from the real reference tensor. Thus, we will first call $gen2s$ on the input content, then call dis_s on the resulting tensor. Thereafter, our loss will be the MSE-Loss between the FC tensor outputted by dis_s and a tensor filled with 0's. Thus, we have the formula below:

$$\mathcal{L}_{dis} = \mathbf{E}\left[(dis\_s(gen2s(content_{ref})) - 0)^2\right]$$

Thus, the total loss for dis_s's optimizer is $\mathcal{L}_{identity} + \mathcal{L}_{dis}$.

2.4 Expriment

2.4.1 Experiment Algorithm

We implemented class `CGANexp` to run training experiments on our generator and discriminator models under the training algorithm mentioned above provided by `CGANTrainer`. This class inherit from class `nt.Experiment`. To initialize a `CGANexp` object, we need to pass in a `CGANTrainer` object, store it as an object attribute `self.trainer`, which contains all models to be trained and their optimizers. We modified the load and save method so that it will be able to recover the states of all networks and optimizers in `self.trainer`. We can also pass in `train_set`, `output_dir`, and etc. as we will do for `nt.Experiment` during initialization.

When calling run on a `CGANexp` object, we will use the methods `train`, `update_lr` to update the models for `num_epochs` epochs.

2.5 Experimental Settings and Results

For our pretrained model for Domain-orientated Cycle-GAN, we used the following data:

Content Train Dataset Landcape pictures from FLickr[4]

Style Train reference Vincent van Gogh artworks from WikiArt[5]

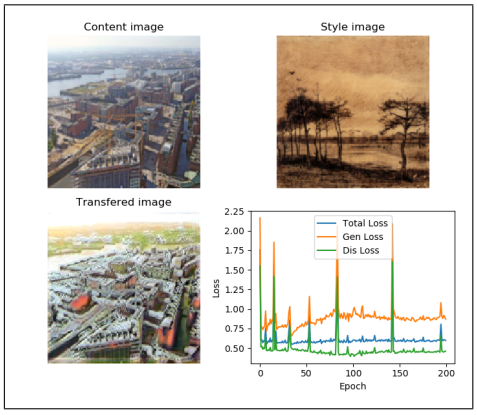
Experimental setting We ran one experiment with different settings:

- **exp_smallBS**: Ran 200 epoch, with each epoch trained on all van Gogh artworks available from content category city, forest, road, batch-size 2, and constant learning rate 1×10^{-3}.
- **exp_largeBS**: Ran 200 epoch, with each epoch trained on all van Gogh artworks available from content category city, forest, road, batch-size 16, and constant learning rate 1×10^{-3}.
- **exp_largeBS_decay**: Ran 200 epoch, with each epoch trained on all van Gogh artworks available from content category city, forest, road, batch-size 16, and learning rate starting from 2×10^{-3} exponentially decaying with a factor of 0.99 every 2 epoch.

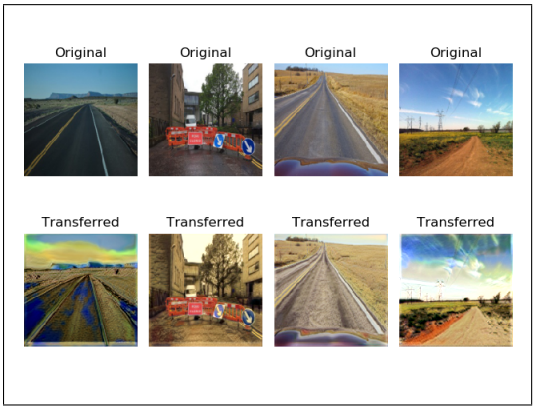
Result The result of each experiment is shown in the figures below. Through comparison, we conclude that running our model with large batch size and learning rate exponentially decay over epochs results in the best result (shown in Figure 4 below).

[4]https://www.flickr.com/groups/landcape/

[5]https://www.wikiart.org/en

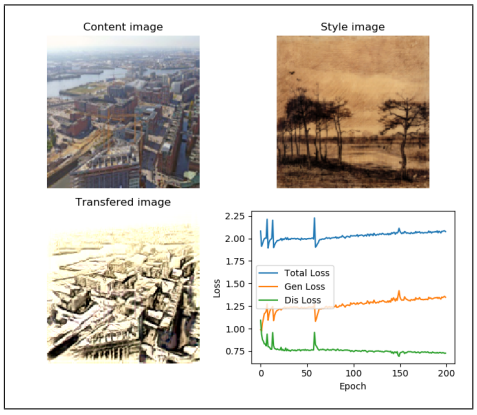


(a) Training Result & Loss Plots

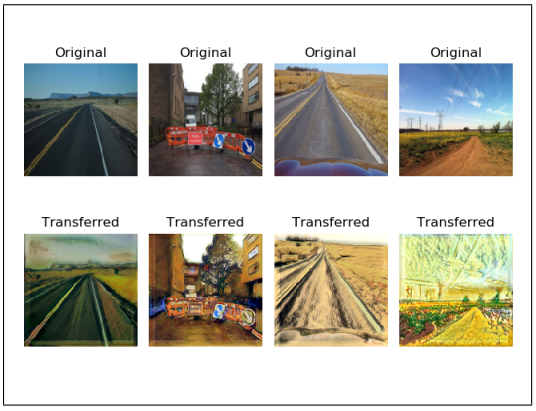


(b) Style Transforming Results

Figure 2: Domain Style Transformation with exp_smallBS

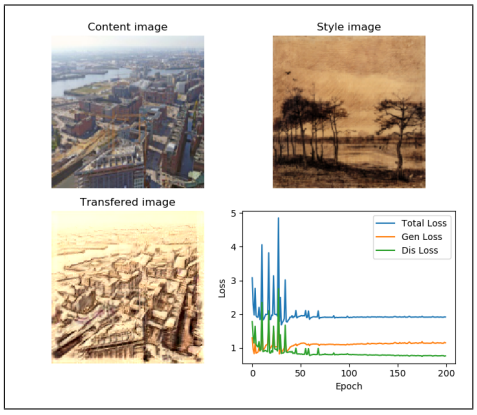


(a) Training Result & Loss Plots

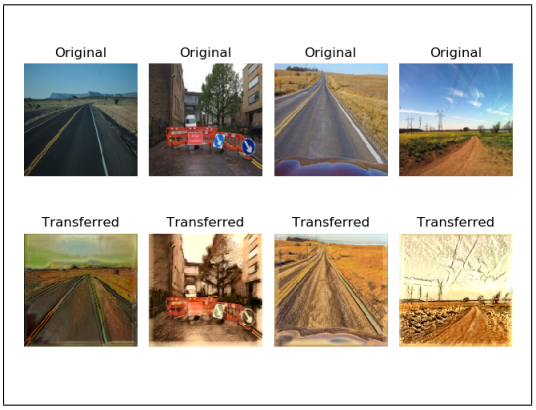


(b) Style Transforming Results

Figure 3: Domain Style Transformation with exp_largeBS



(a) Training Result & Loss Plots



(b) Style Transforming Results

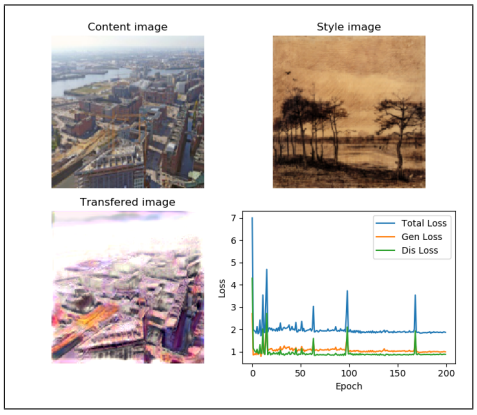
Figure 4: Domain Style Transformation with exp_largeBS_decay

2.6 Challenges and Failure cases

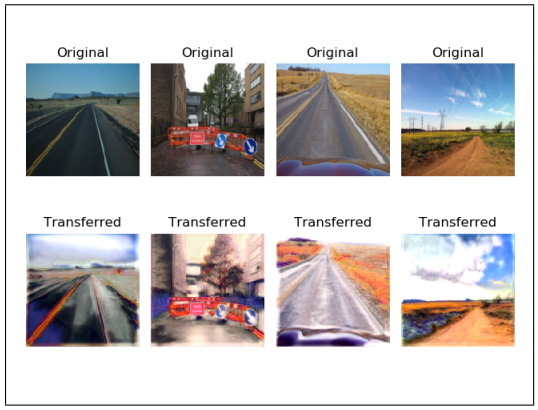
Challenges One of the main challenges we met while training the model is the inconsistency between style pictures and content pictures. Specifically, they are very different in both size and resolution. Therefore, we have to crop and resize them into same size for training. However, cropping causes us lose information on the style and resizing causes us have smaller tensor to train on.

Also, the robustness of our model depends on the variety of style & content pictures provided to us. Currently, there is only about 1300 pictures of van Gogh available. Thus, it limit our model's generality to some extend. We tried to solve this challenge through pairing a style picture with multiple content pictures for training to enlarge our training set.

Failure Case One of the failure case we encountered is when we try to exponentially decay the learning rate of our optimizer with small batch-size. The failure plots are shown below.



(a) Training Result & Loss Plots



(b) Style Transforming Results

Figure 5: Domain Style Transformation with exp_smallBS_decay

Acknowledgments

Use unnumbered third level headings for the acknowledgments. All acknowledgments go at the end of the paper. Do not include acknowledgments in the anonymized submission, only in the final paper.

References

[1] L. Gatys, A. Ecker, and M. Bethge. A neural algorithm of artistic style. Nature Communications, 2015.

[2] L. Gatys, A. Ecker, and M. Bethge. Image Style Transfer Using Convolutional Neural Networks, 2016.

[3] Alexander, J.A. & Mozer, M.C. (1995) Template-based algorithms for connectionist rule extraction. In G. Tesauro, D.S. Touretzky and T.K. Leen (eds.), *Advances in Neural Information Processing Systems 7*, pp. 609–616. Cambridge, MA: MIT Press.

[4] Bower, J.M. & Beeman, D. (1995) *The Book of GENESIS: Exploring Realistic Neural Models with the GEneral NEural SImulation System.* New York: TELOS/Springer–Verlag.

[5] Hasselmo, M.E., Schnell, E. & Barkai, E. (1995) Dynamics of learning and recall at excitatory recurrent synapses and cholinergic modulation in rat hippocampal region CA3. *Journal of Neuroscience* **15**(7):5249-5262.