Lab: CudaVision – Learning Vision Systems on Graphics Cards (MA-INF 4308)

Assignment 6

4.7.2016

Prof. Sven Behnke
Dr. Seongyong Koo

universitätbonn ais

Denoising Autoencoders (DAE)

- **Goal: Implementing DAE to denoise MNIST data**

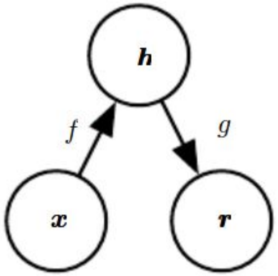
- **Autoencoders**

- **Denoising Autoencoders**
  - Fully connected hidden layers
  - Convolutional Denoising Autoencoders

- **Theoretical reference**
  - http://www.deeplearningbook.org/contents/autoencoders.html
  - P. Vincent, H. Larochelle Y. Bengio and P.A. Manzagol, Extracting and Composing Robust Features with Denoising Autoencoders, Proceedings of the Twenty-fifth International Conference on Machine Learning (ICML‘08), pages 1096 - 1103, ACM, 2008.

Autoencoders

- An autoencoder is a neural network that is trained to attempt to copy its input to its output.
  - Encoder function $\boldsymbol{h} = f(\boldsymbol{x})$
  - Decoder function $\boldsymbol{r} = g(\boldsymbol{h})$



- Autoencoders are designed to copy only input approximately that resembles the training data. --> it often learns useful properties of the data, because the model is forced to prioritize which aspects of the input should be copied
  - constrain *h* to have smaller dimension than *x*

- Learning process: minimizing $L(\boldsymbol{x}, g(f(\boldsymbol{x})))$, e.g. mean squared error

- If *f* and *g* are linear and L is MSE, encoder works same as PCA
- With nonlinear *f* and *g* functions, more powerful nonlinear generalized PCA
  - but, if the learning capacity is too great, the copying task can fail to learn anything useful about the dataset

Denoising Autoencoders

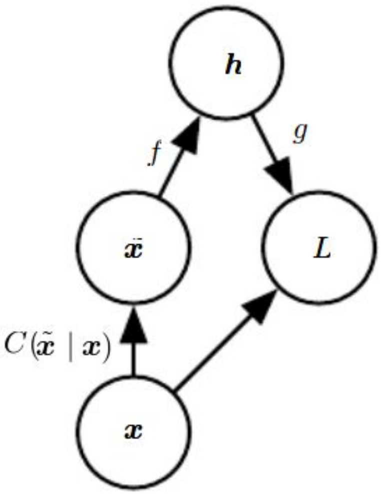
- We can obtain an autoencoder that learns something useful by changing the reconstruction error term of the cost function

- Denoising Autoencoders (DAE) minimizes $L(\boldsymbol{x}, g(f(\tilde{\boldsymbol{x}})))$, where $\tilde{\boldsymbol{x}}$ is a copy of $\boldsymbol{x}$ that has been corrupted by some form of noise.

- Denoising autoencoders must be trained to undo this corruption
  - Corruption process $C(\tilde{\mathbf{x}} \mid \mathbf{x})$: conditional distribution over corrupted samples, given a data sample
  - Training process is to learn a reconstruction distribution $p_{\text{reconstruct}}(\mathbf{x} \mid \tilde{\mathbf{x}})$ estimated from training pairs $(\boldsymbol{x}, \tilde{\boldsymbol{x}})$

    $$p_{\text{reconstruct}}(\boldsymbol{x} \mid \tilde{\boldsymbol{x}}) = p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$$

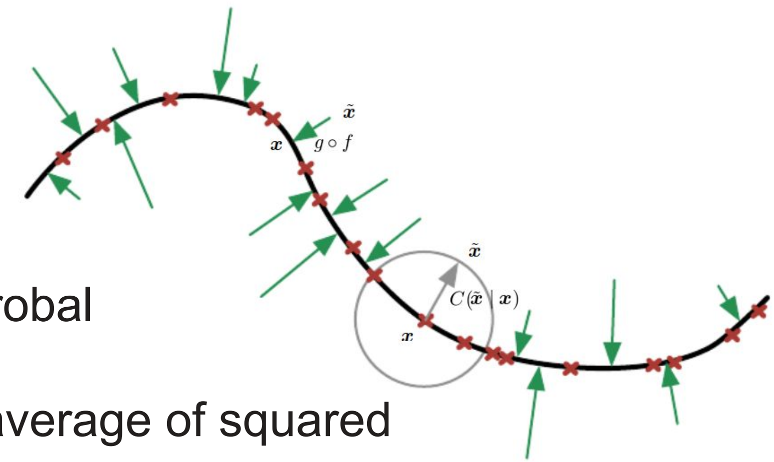
  - Gradient-based approximate minimization on the negative log-likelihood $-\log p_{\text{decoder}}(\boldsymbol{x} \mid \boldsymbol{h})$



universitätbonn ais

Denoising Autoencoders

- DAEs with Gaussian corrumption $C(\tilde{\mathbf{x}} = \tilde{\boldsymbol{x}}|\boldsymbol{x}) = \mathcal{N}(\tilde{\boldsymbol{x}}; \mu = \boldsymbol{x}, \Sigma = \sigma^2 I)$ makes the autoencoder learn a vector field $(g(f(\boldsymbol{x})) - \boldsymbol{x})$ that estimates the score of the data distribution.



  - map a corrupted data point back to the original data point (red cross)
  - In the corruption process $C(\tilde{\mathbf{x}} \mid \mathbf{x})$ (gray circle): one training example is transformed into one sample in equiprobal gray circle
  - In the training process: minimize the average of squared errors $||g(f(\tilde{\boldsymbol{x}})) - \boldsymbol{x}||^2$
  - After training, the vector $g(f(\tilde{\boldsymbol{x}})) - \tilde{\boldsymbol{x}}$ points approximately towards the nearest point on the manifold, since $g(f(\tilde{\boldsymbol{x}}))$ estimates the center of mass of the clean points $\boldsymbol{x}$ which could have given rise to $\tilde{\boldsymbol{x}}$.
  - The autoencoder thus learns a vector field $(g(f(\boldsymbol{x})) - \boldsymbol{x})$ indicated by the green arrows.

universitätbonn ais

DAE Example

- In `dae_mnist.ipynb`
- Construct DAE network with two fully connected hidden layers

```
# Network Parameters
n_hidden_1 = 256 # 1st layer num features
n_hidden_2 = 256 # 2nd layer num features
n_input    = 784 # MNIST data input (img shape: 28*28)
n_output   = 784 #

# tf Graph input
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_output])
dropout_keep_prob = tf.placeholder("float")

h1_w = tf.Variable(tf.random_normal([n_input, n_hidden_1]))
h2_w = tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2]))
out_w = tf.Variable(tf.random_normal([n_hidden_2, n_output]))
h1_b = tf.Variable(tf.random_normal([n_hidden_1]))
h2_b = tf.Variable(tf.random_normal([n_hidden_2]))
out_b = tf.Variable(tf.random_normal([n_output]))

layer_1 = tf.nn.sigmoid(tf.add(tf.matmul(x, h1_w), h1_b))
layer_1out = tf.nn.dropout(layer_1, dropout_keep_prob)
layer_2 = tf.nn.sigmoid(tf.add(tf.matmul(layer_1out, h2_w), h2_b))
layer_2out = tf.nn.dropout(layer_2, dropout_keep_prob)
out = tf.nn.sigmoid(tf.matmul(layer_2out, out_w) + out_b)
cost = tf.reduce_mean(tf.pow(out-y, 2))
```

DAE Example

- In `dae_mnist.ipynb`

- Training with MNIST data
    - 28 x 28 image as Input (x)
    - 28 x 28 corrupted image with Gaussian noise as output (y)

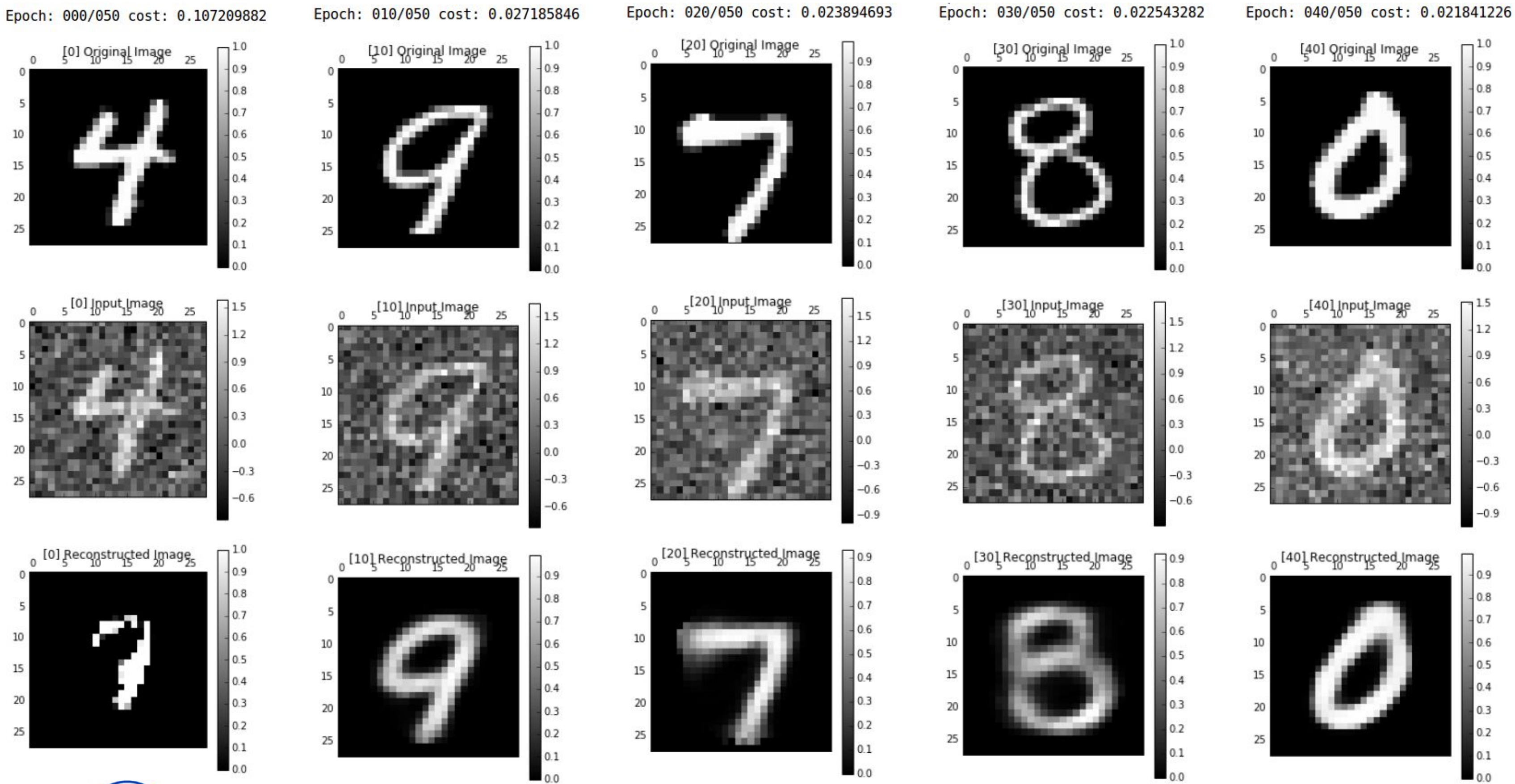
```
for i in range(num_batch):
    randidx = np.random.randint(trainimg.shape[0], size=batch_size)
    batch_xs = trainimg[randidx, :]
    batch_xs_noisy = batch_xs + 0.3*np.random.randn(batch_xs.shape[0], 784)

    batch_ys = trainlabel[randidx, :]

    # Fit training using batch data
    sess.run(optimizer, feed_dict={x: batch_xs_noisy, y: batch_xs, dropout_keep_prob: 0.5})
    # Compute average loss
    avg_cost += sess.run(cost, feed_dict={x: batch_xs_noisy, y: batch_xs, dropout_keep_prob: 1})/num_batch
```

DAE Example

- In `dae_mnist.ipynb`
- Test result for each training epoch

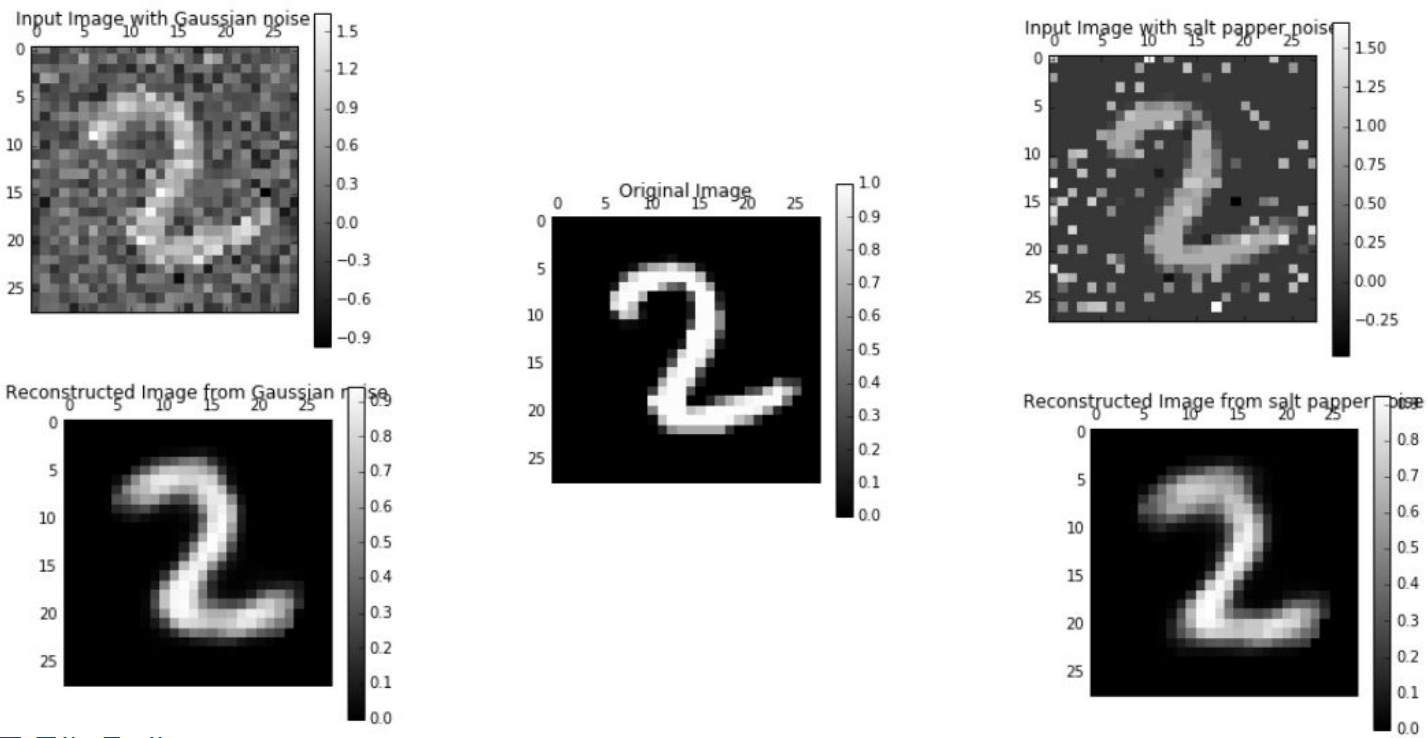


DAE Example

- Test with another corrupted image with different noise

```
print ("Gaussian Noise")
noisyvec_gauss = testvec + 0.3*np.random.randn(1, 784)

print ("Salt and Pepper Noise")
noisyvec_saltpapper = testvec
rate      = 0.15
noiseidx = np.random.randint(testimg.shape[1], size=int(testimg.shape[1]*rate))
noisyvec_saltpapper[0, noiseidx] = 1-noisyvec[0, noiseidx]
```



Assignment 6

- **Convolutional Autoencoder (CAE)**
  - Zeiler, Matthew D and Krishnan, Dilip and Taylor, Graham W and Fergus, Rob, "Deconvolutional Networks", CVPR 2010

- **Construct Convolutional Autoencoder for denoising images**
  - Desing encoders and decoders with convolutional layers
    e.g. 3 ConvLayers (1xn1, n1xn2, n2xn3) for encoders,
    3 ConvLayers (n3xn2, n2xn1, n1x1) for decoders
  - You may need to use `tf.pack` to make a output_shape of `tf.nn.conv2d_transpose` for decoders

- **Training Convolutional Autoencoders**
  - with a MSE cost function to minimize $L(\boldsymbol{x}, g(f(\boldsymbol{x})))$
  - Reconstruct images from corrupted images with Gaussian noise
    - Compare the result with DAE
  - Reconstruct images from corrumpted images by normalization
    - Compare the result with DAE

[How to normalize images in dataset]

```
mean_img = np.mean(mnist.train.images, axis=0)
test_xs, _ = mnist.test.next_batch(n_examples)
test_xs_norm = np.array([img - mean_img for img in test_xs])
```