Tutorial on autoencoders

Rafael Baeta

Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

18 de novembro de 2016

DCC
DEPARTAMENTO DE
CIÊNCIA DA COMPUTAÇÃO

UFMG

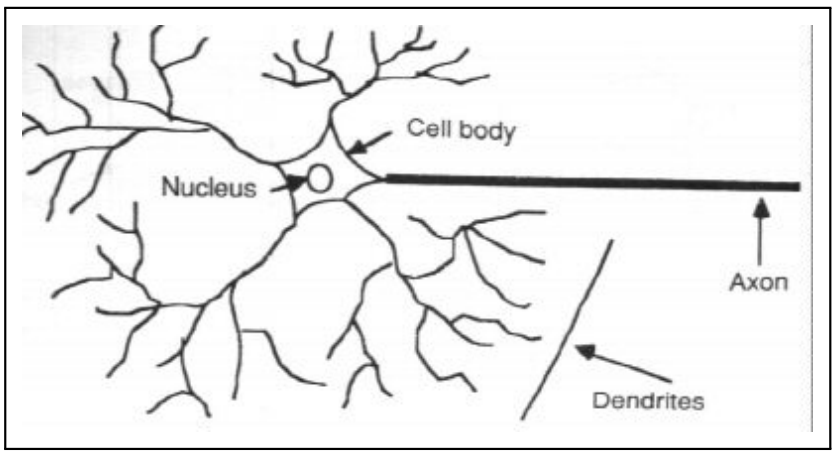
Introduction

- Autoencoder is a type of neural network.
- Its considered an algorithm of unsupervised learning
- Its aim is to generate representative features of the input, compress images and some autoencoders can genenerate new images from the original
- Perhaps, the first work to study unsupervisied learning with autoencoder was published in 1987.[1]

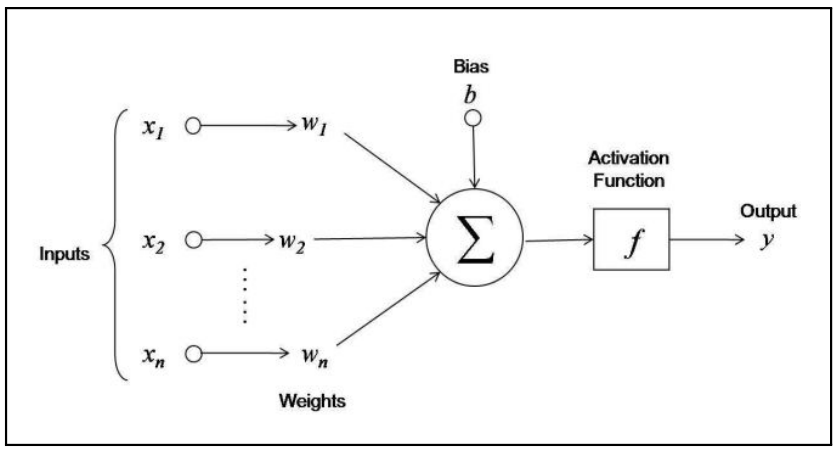
[1] Dana H Ballard. "Modular Learning in Neural Networks." Em: *AAAI*. 1987, pp. 279–284.

Introduction

- An autoencoder as others neural networks as constituted by artificial neurons
- A artificial neuron tries to mimic a real neuron.
- It's constituted by three parts: a set of inputs, an activation function and a output.



Neuron



Artificial neuron

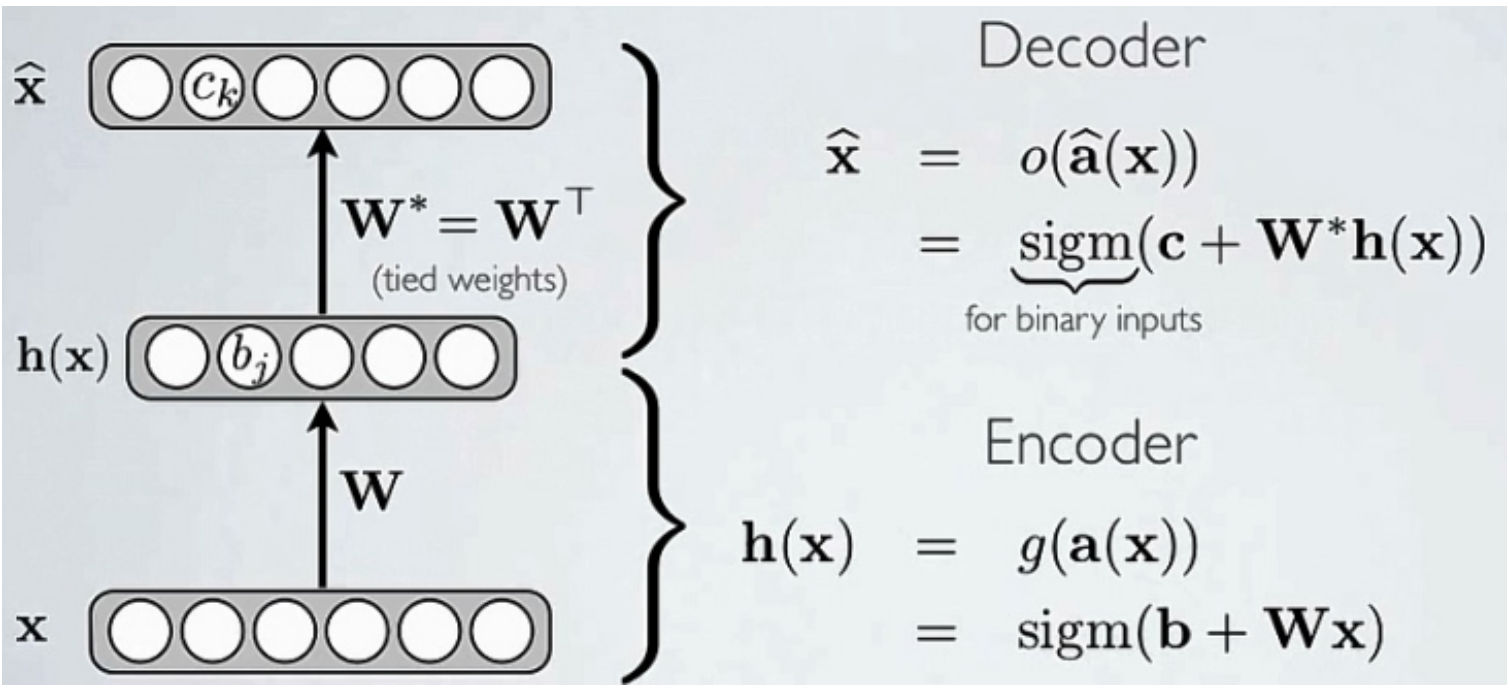
Introduction

- The organization of neurons generates different types of neural networks.

- MLP (Multilayer Perceptron)
- AE (Auto encoder)
- RBM (Restricted boltzmann machine)
- RBF (Radial basis function)
- CNN (Convolutional Neural Network)
- RNN (Recurrent Neural Network)

Autoencoder

- Try to reproduce its input in the output layer and h(x) is a latent representation of the input.
  - Output layer has the same size of the input layer



Autoencoder

Loss function

- We need something to optimize
  - How well the network reproduces the inputs.
    - Stochastic optimization with gradient descent
- Loss for binary inputs:
  - $L(W) = -\sum_k (x_k log(\hat{x}_k) + (1 - x_k) log(1 - \hat{x}_k))$
    - Cross-entropy: it minimizes if $x_i = \hat{x}_i$
- Loss for real-valued inputs:
  - $L(W) = \frac{1}{2} \sum_k (\hat{x}_k = x_k)^2$
    - Sum of squared differences
- For both cases, the gradient $\nabla_W L$ has a very simple form:
  - $\nabla_W L = \hat{x}^{(t)} - x^{(t)}$

Autoencoder

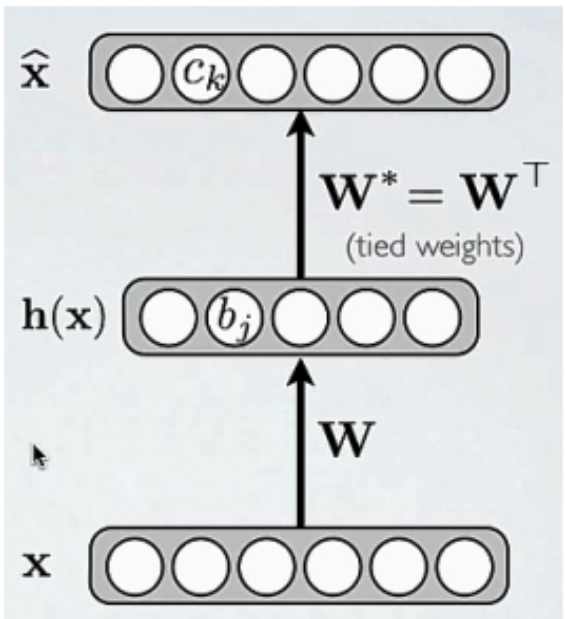
Undercomplete and overcomplete hidden layers.

- Autoencoders are learned to create compressed representation of information that is coming from inputs.
  - Compression here is considered in sense of compressed with lost of information", that is, autoencoders extract the most important information and convert it to new compressed, less dimensional form (i.e. representation).
  - What is the impact of the size of the hidden layer? Impact in terms of compression, that is, the types of features the autoencoder will learn

Autoencoder

Undercomplete representation.

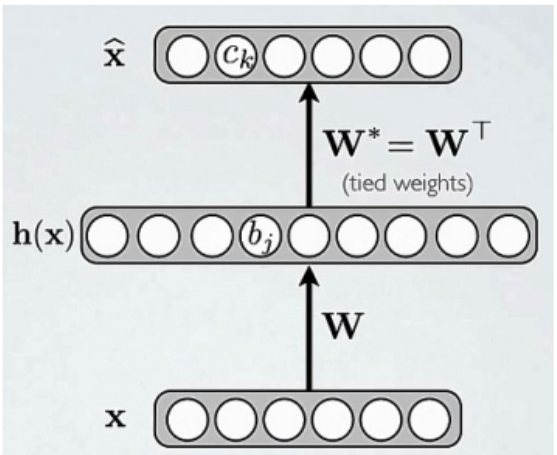
- Hidden layer is smaller than the input layer.
  - In this case, hidden layer compresses the input
  - It is trained to compress well only inputs that are generated following the training distribution
  - The representation works well for objects like the ones in the training set



Autoencoder

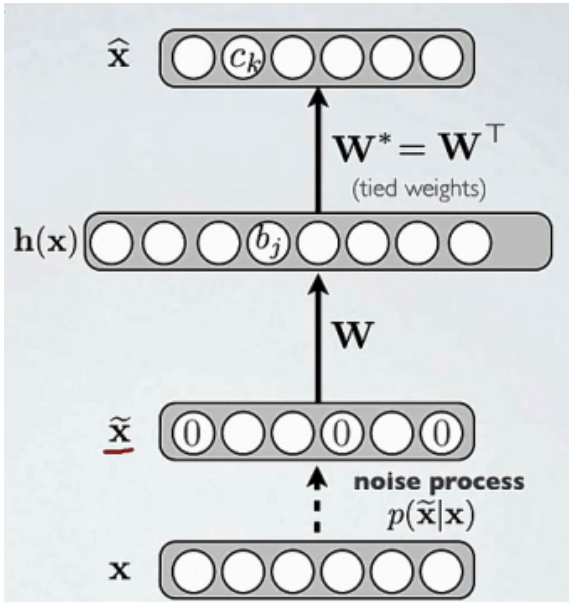
Overcomplete representation.

- Hidden layer is larger than the input layer.
  - In this case, hidden layer does not compress the input
    - No guarantee that the hidden units will extract a meaningful structure from the inputs
    - Each hidden unit tend to simply copy a different input component.



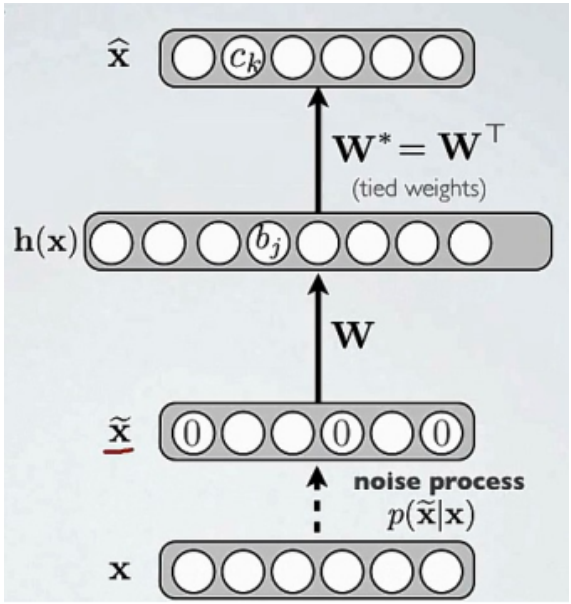
Autoencoder - Denoising Autoencoder

- Prevent (overcomplete) autoencoder to copy the input.
  - Random assignment of inputs to 0, with probability v.
  - Pepper and salt noise.
  - Gaussian additive noise
  - Rescale, rotate, shift.



Autoencoder - Denoising Autoencoder

- Reconstruction of x is computed from $\tilde{x}$
- But, the loss compare the reconstruction ($\hat{x}$) with the original input (x).



Autoencoder - Contractive Autoencoder

- Try to avoid uninteresting representations.
- Add an explicit term in the loss that penalizes these solutions
  - Penalty similarity of hidden and input layers
- Minimize reconstruction error + penalty

Implementation

- We choose Torch.[2]
- The packages mnist, dpnn, rnn (Ex: luarock install mnist)
- Download https://github.com/Kaixhin/Autoencoders/archive/master.zip. It's contains autoencoder models (AE, DAE, VAE).
- Just execute the command "th main.lua -model <modelName>" Ex: "th main.lua -model AE".

[2] *Torch7 Scientific computing for Lua(JIT)*. . http://torch.ch/.

Implementation

- We need Torch.[3]
- The packages mnist, dpnn, rnn (Ex: luarock install mnist)
- Download https://github.com/Kaixhin/Autoencoders/archive/master.zip. It's contains autoencoder models (AE, DAE, VAE).
- Just execute the command "th main.lua -model <modelName>" Ex: "th main.lua -model AE".

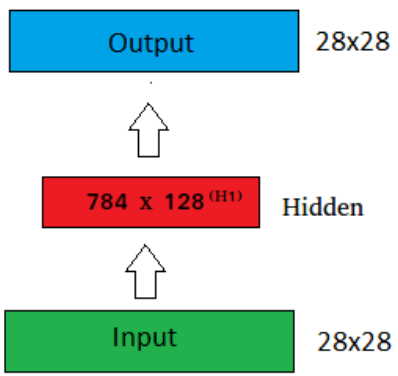
[3] *Torch7 Scientific computing for Lua(JIT)*.

Implementation

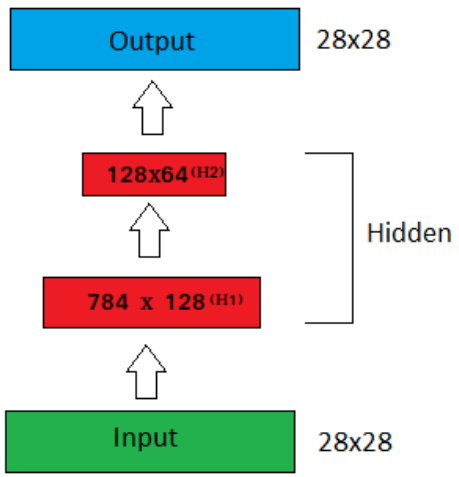
- 60000 (28x28) images and 10 classes
- Minibatches - 150 images
- Epoch - 400
- Learning rate - 0.01
- Adagrad
- BCE(binary cross entropy for sigmoid) criterion

Implementation

- We want an autoencoder with two layer (stacked autoencoder)



Autoencoder block



Desired autoencoder

Implementation

- The first block is trained to reconstruct the raw image.

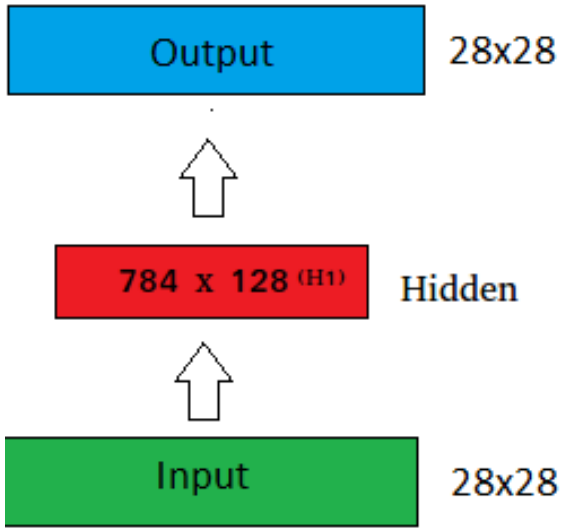


Figura : Autoencoder block

Implementation

- The first block is trained to reconstruct the raw image.
- After the block train we remove the reconstruction layer

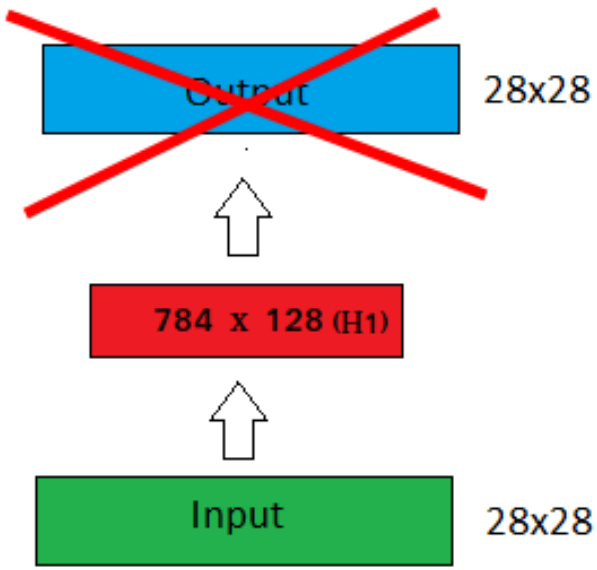


Figura : Autoencoder block

Implementation

- The first block is trained to reconstruct the raw image.
- After the block train we remove the reconstruction layer
- Then, the features of the previous layer is used as input to train the next block.

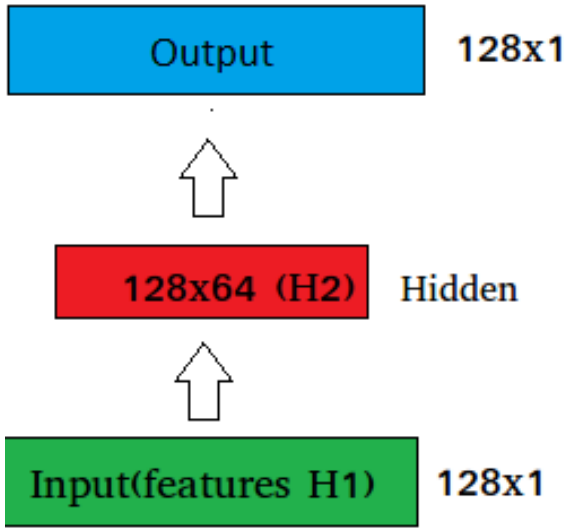


Figura : Autoencoder block

Implementation

- The first block is trained to reconstruct the raw image.
- After the block train we remove the reconstruction layer
- Then, the features of the previous layer is used as input to train the next block.
- This process can be repeated how times as necessary

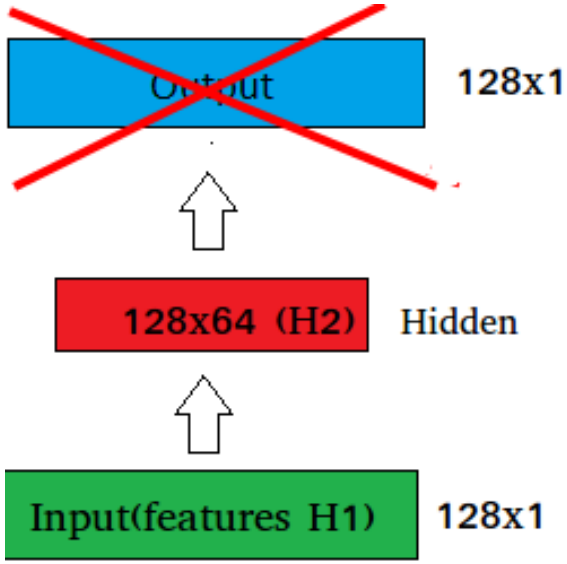


Figura : Autoencoder block

Implementation

- Now, we need to build the stacked autoencoder.

Implementation

- Now, we need to build the stacked autoencoder.
- Put the original input layer

Figura : Autoencoder block

Implementation

- Now, we need to build the stacked autoencoder.
- Put the original input layer
- Put the first feature layer

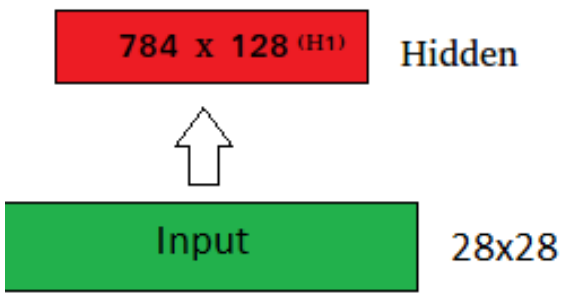


Figura : Autoencoder block

Implementation

- Now, we need to build the stacked autoencoder.
- Put the original input layer
- Put the first feature layer with the trained weights
- Put the second feature layer with the trained weights

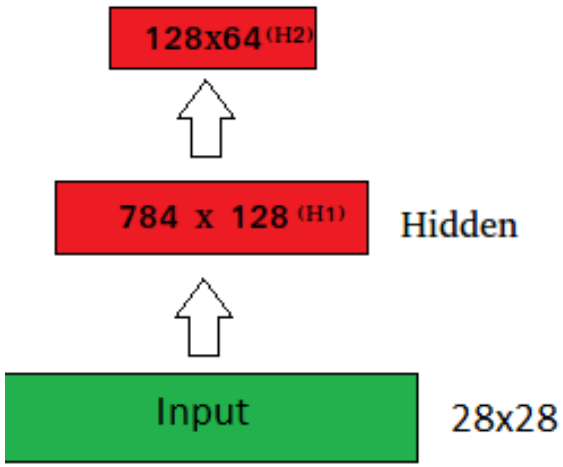


Figura : Autoencoder block

Implementation

- Now, we need to build the stacked autoencoder.
- Put the original input layer
- Put the first feature layer with the trained weights
- Put the second feature layer with the trained weights
- Finally, the decoder (output) must be inserted and trained to reconstruct.

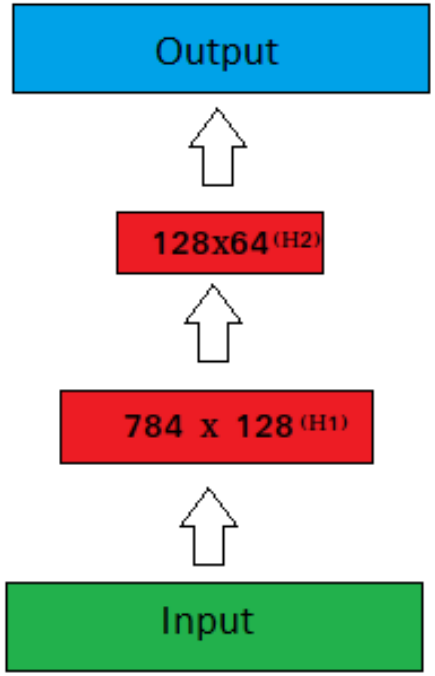


Figura : Autoencoder block

Implementation

- Now, we need to build the stacked autoencoder.
- Put the original input layer
- Put the first feature layer with the trained weights
- Put the second feature layer with the trained weights
- Finally, the decoder (output) must be inserted and trained to reconstruct.

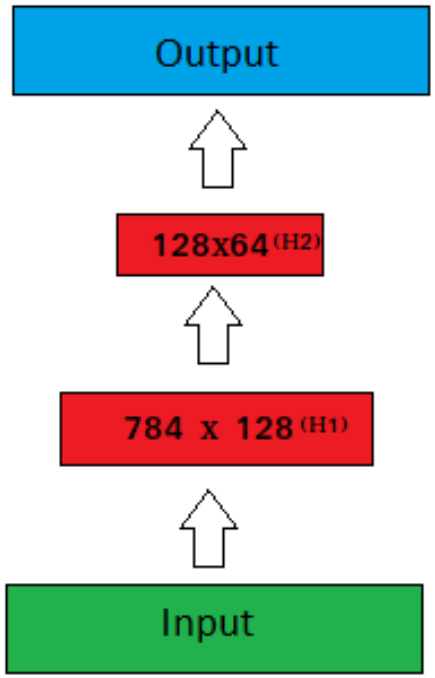


Figura : Autoencoder block

Implementation

```
1: local Model = {}
2: function Model:createAutoencoder(height,width,neurons,output1,output2)
3: local featureSize = height * width
4: – Create encoder
5: self.encoder = nn.Sequential()
6: self.encoder:add(nn.View(-1, featureSize))
7: self.encoder:add(nn.Linear(featureSize, neurons))
8: self.encoder:add(nn.ReLU(true))
9: – Create encoder
10: self.decoder = nn.Sequential()
11: self.decoder:add(nn.Linear(neurons,height*width))
12: self.decoder:add(nn.Sigmoid(true))
13: self.decoder:add(nn.View(output1,output2))
14: – Create autoencoder
15: self.autoencoder = nn.Sequential()
16: self.autoencoder:add(self.encoder)
17: end
```

Algorithm 1: Autoencoder model

Implementation


```
1:  if cuda then then
2:      autoencoder:cuda()
3:  end if
4:  – Get parameters
5:  local theta, gradTheta = autoencoder:getParameters()
6:  local epoch
7:  local SAE
8:  – Create loss
9:  local criterion = nn.BCECriterion()
10: if cuda then then
11:     criterion:cuda()
12: end if
13: – Create optimiser function evaluation
14: local x – Minibatch
15: initialNeurons = 128
```

Algorithm 2: Training

Implementation


```
1:  for numEncoder = 1,opt.encoders do
2:      local __, loss
3:      local optimParams = {learningRate = opt.learningRate}
4:      autoencoder:training()
5:      theta, gradTheta = autoencoder:getParameters()
6:      local feval = function(params)
7:      if theta  = params then then
8:          theta:copy(params)
9:      end if
10:     gradTheta:zero()
11:     – Reconstruction phase
12:     – Forward propagation
13:     local xHat = autoencoder:forward(x) – Reconstruction
14:     local loss = criterion:forward(xHat, x)
15:     – Backpropagation
16:     local gradLoss = criterion:backward(xHat, x)
17:     autoencoder:backward(x, gradLoss)
18:     return loss, gradTheta
19:     end
20: end for
```

Algorithm 3: Training

Implementation

```
for numEncoder = 1,opt.encoders do
    local losses = {}
    latent_data = {
    data = torch.CudaTensor(XTrain:size(1),initialNeurons,1),
    size = function() return XTrain:size(1) end
    }
    for epoch = 1, opt.epochs do
        print('Epoch ' .. epoch .. '/' .. opt.epochs)
        cont = 1
        for n = 1, N, opt.batchSize do
            -- Get minibatch
            if numEncoder == 1 or numEncoder == opt.encoders then then
                x = XTrain:narrow(1, n,opt.batchSize)
            else
                x = latents:narrow(1,n,opt.batchSize)
            end if
            -- Optimise
            __, loss = optim[opt.optimiser](feval, theta, optimParams)
        end for
    end for
end for
```

Algorithm 4: Training

Implementation


```
1:  for numEncoder = 1,opt.encoders do
2:      for epoch = 1, opt.epochs do
3:          print('Epoch ' .. epoch .. '/' .. opt.epochs)
4:          cont = 1
5:          for n = 1, N, opt.batchSize do
6:              if epoch == opt.epochs and numEncoder < (opt.encoders-1) then
7:                  latent = autoencoder:get(1).output
8:                  for i = 1, opt.batchSize do
9:                      latent_data.data[cont] = latent[i]
10:                     cont = cont + 1
11:                 end for
12:             end if
13:             losses[#losses + 1] = loss[1]
14:         end for
15:     end for
16: end for
```

Algorithm 5: Training

Implementation

```
1: for numEncoder = 1,opt.encoders do
2:     if numEncoder == 1 then
3:         encoder = autoencoder
4:         encoder:remove(2)
5:     else if numEncoder > 1 and numEncoder < (opt.encoders-1) then
6:         autoencoder:remove(2)
7:         encoder:get(1):insert(autoencoder:get(1):get(2))
8:         encoder:get(1):insert(autoencoder:get(1):get(3))
9:     else if numEncoder == (opt.encoders-1) then
10:        autoencoder:remove(2)
11:        encoder:get(1):insert(autoencoder:get(1):get(2))
12:        encoder:get(1):insert(autoencoder:get(1):get(3))
13:        decoder = nn.Sequential()
14:        :add(nn.Linear(initialNeurons,784))
15:        :add(nn.Sigmoid(true))
16:        :add(nn.View(28,28))
17:        SAE = encoder
18:        SAE:add(decoder)
19:        autoencoder = SAE
20:        autoencoder:cuda()
21:    end if
22: end for
```

Algorithm 6: Training

Implementation


```
1:  for numEncoder = 1,opt.encoders do
2:      if numEncoder < (opt.encoders-1) then
3:          latents = latent_data.data:float()
4:          latents = latents:cuda()
5:          Model:createAutoencoder(initialNeurons,1,initialNeurons/2,initialNeurons,1)
6:          autoencoder = Model.autoencoder
7:          autoencoder:cuda()
8:          initialNeurons = initialNeurons/2
9:      end if
10: end for
```

Algorithm 7: Training

Implementation

```
1: print('Testing')
2: x = XTest:narrow(1, 1, 10)
3: local xHat
4: autoencoder:evaluate()
5: xHat = autoencoder:forward(x)
```

Algorithm 8: Testing

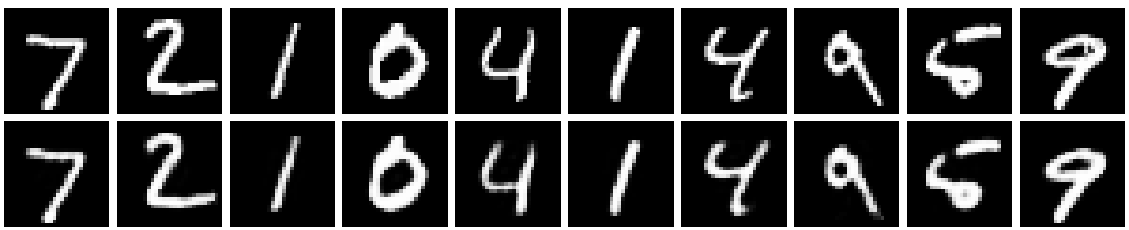


Figura : Reconstruction

Conclusion

- Autoencoders can reduce images better than PCA.
- Its can generate new features.
- Exist many types of autoencoders that can be used for different tasks

References I

Ballard, Dana H. “Modular Learning in Neural Networks.” Em: *AAAI*. 1987, pp. 279–284.

Le, Quoc V. *A Tutorial on Deep Learning Part 2: Autoencoders, Convolutional Neural Networks and Recurrent Neural Networks*. 2015.

Torch implementations of various types of autoencoders. https://github.com/Kaixhin/Autoencoders.

Torch7 Scientific computing for Lua(JIT). http://torch.ch/.

UFLDL Tutorial - Autoencoder. http://ufldl.stanford.edu/tutorial/unsupervised/Autoencoders/.