Autoencoders

Sargur Srihari
srihari@buffalo.edu

Topics

- What is an autoencoder?

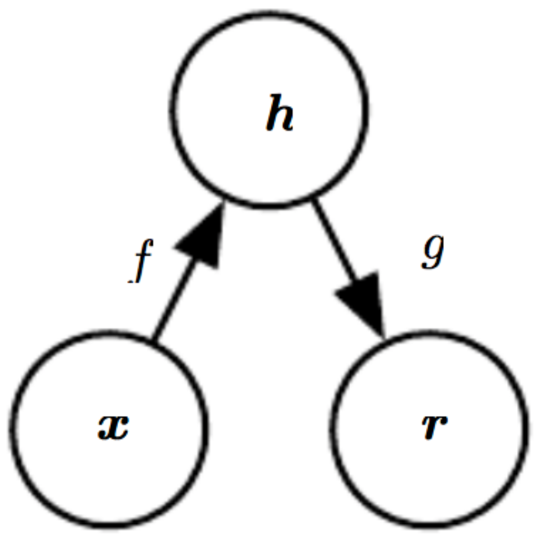
1. Undercomplete Autoencoders
2. Regularized Autoencoders
3. Representational Power, Layout Size and Depth
4. Stochastic Encoders and Decoders
5. Denoising Autoencoders
6. Learning Manifolds and Autoencoders
7. Contractive Autoencoders
8. Predictive Sparse Decomposition
9. Applications of Autoencoders

What is an Autoencoder?

- It is an artificial neural network that is trained to attempt to copy its input to its output
- It has a hidden layer $\boldsymbol{h}$ that describes the code used to represent the input

General structure of an autoencoder

- Maps an input $\boldsymbol{x}$ to an output $\boldsymbol{r}$ (called reconstruction) through an internal representation code $\boldsymbol{h}$
  - It has a hidden layer $\boldsymbol{h}$ that describes a code used to represent the input
- The network has two parts
  - The encoder function $\boldsymbol{h}=f(\boldsymbol{x})$
  - A decoder that produces a reconstruction $\boldsymbol{r}=g(\boldsymbol{h})$

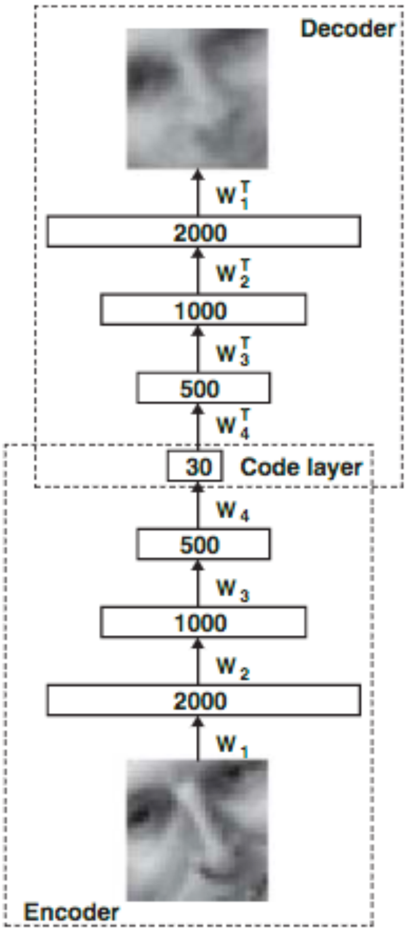


Rationale of an Autoencoder

- An autoencoder that simply learns to set $g\,(f\,(\boldsymbol{x}))=\boldsymbol{x}$ everywhere is not especially useful
- Autoencoders are designed to be unable to copy perfectly
  - They are restricted in ways to copy only approximately
  - Copy only input that resembles training data
- Because model is forced to prioritize which aspects of input should be copied, it often learns useful properties of the data
- Modern autoencoders have generalized the idea od encoder and decoder beyond deterministic functions to stochastic mappings $p_{\text{encoder}}(\boldsymbol{h}|\boldsymbol{x})$ and $p_{\text{decoder}}(\boldsymbol{x}|\boldsymbol{h})$

Use of Autoencoders

1. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction
2. It is now widely used for learning generative models of data



Undercomplete Autoencoders

- Copying input to output sounds useless
- Instead we hope that training the autoencoder will result in $\boldsymbol{h}$ taking on useful properties
- One way to obtain useful features is to constrain $\boldsymbol{h}$ to have a smaller dimension than $\boldsymbol{x}$
  - This is called *undercomplete*
  - It forces the autoencoder to capture the most salient features of the training data

Autoencoder History

- Part of neural network landscape for decades
- Traditionally used for dimensionality reduction and feature learning
- Connection between autoencoders and latent variable models have brought them into forefront of generative models

Autoencoder Training

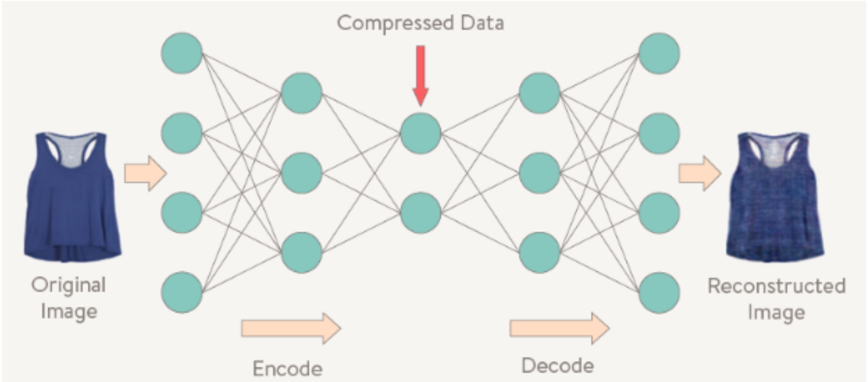
- An autoencoder is a feed-forward non-recurrent neural net which is very similar to an MLP
  - With an input layer, an output layer and one or more hidden layers
- Can be trained using the same techniques
  - Compute gradients using back-propagation
  - Followed by minibatch gradient descent
- Unlike feedforward networks, can be trained using *Recirculation*
  - Compare activations on the input to activations of the reconstructed input
  - More biologically plausible than back-prop but rarely used in ML

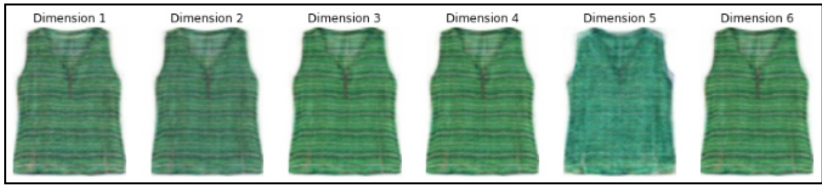
1. Undercomplete Autoencoder

- Copying input to output sounds useless
- But we are not interested in the output of the decoder
- We hope that training the autoencoder to perform copying task will result in $\boldsymbol{h}$ taking on useful properties
- To obtain useful features, constrain $\boldsymbol{h}$ to have lower dimension than $\boldsymbol{x}$
- Such an autoencoder is called undercomplete
- Learning the undercomplete representation forces the autoencoder to capture most salient features of training data

Deepstyle

- Boil down to a representation which relates to style
    - By iterating neural network through a set of images learn efficient representations
- Choosing a random numerical description in encoded space will generate new images of styles not seen
- Using one input image and changing values along different dimensions of feature space you can see how the generated image changes (patterning, color texture) in style space





Autoencoder with linear decoder +MSE is PCA

- Learning process is that of minimizing a loss function

$$L(\boldsymbol{x}, g(f(\boldsymbol{x})))$$

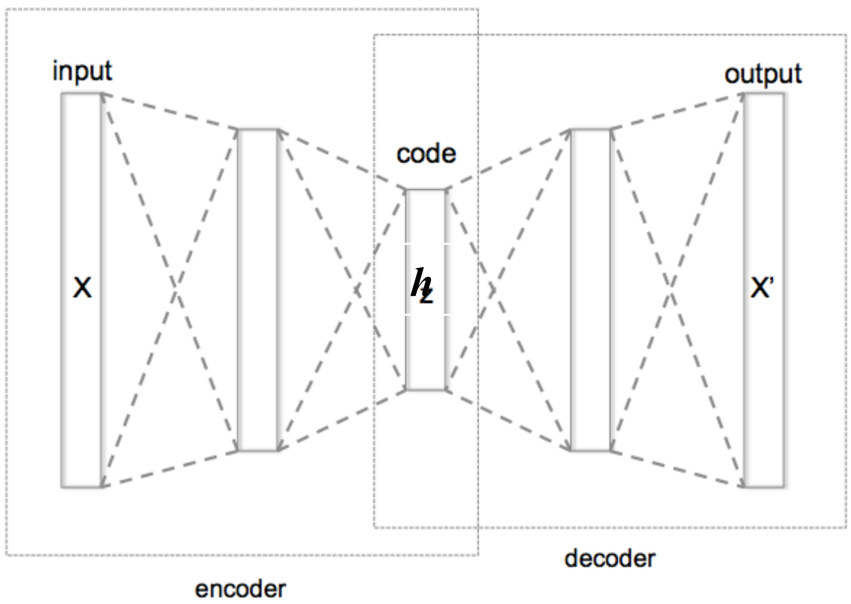
  - where L is a loss function penalizing $g(f(\boldsymbol{x}))$ for being dissimilar from $\boldsymbol{x}$, such as L^2 norm of difference: mean squared error
- When the decoder is linear and L is the mean squared error, an undercomplete autoencoder learns to span the same subspace as PCA
- In this case the autoencoder trained to perform the copying task has learned the principal subspace of the training data as a side-effect
- Autoencoders with nonlinear f and g can learn more powerful nonlinear generalizations of PCA

Autoencoder structure

- Encoder f and decoder g

$$f : X \rightarrow \boldsymbol{h}$$
$$g : \boldsymbol{h} \rightarrow X$$
$$\arg\min_{f,g} \left\| X - (f \circ g) X \right\|^2$$

Autoencoder with 3 fully connected hidden layers



- One hidden layer
  - Non-linear encoder
  - Takes input $\boldsymbol{x}\ \varepsilon\ R^d$
  - Maps into output $\boldsymbol{h}\ \varepsilon\ R^p$

$$\boldsymbol{h} = \sigma_1(W\boldsymbol{x} + \boldsymbol{b})$$
$$\boldsymbol{x}' = \sigma_2(W'\boldsymbol{h} + \boldsymbol{b}')$$

σ is an element-wise activation function such as sigmoid or Relu

Trained to minimize reconstruction error (such as sum of squared errors)

$$L(\boldsymbol{x},\boldsymbol{x}') = \left\|\boldsymbol{x} - \boldsymbol{x}'\right\|^2 = \left\|\boldsymbol{x} - \sigma_2(W^t(\sigma_1(W\boldsymbol{x} + \boldsymbol{b})) + \boldsymbol{b}')\right\|^2$$

Provides a compressed representation of the input x

Encoder/Decoder Capacity

- If encoder f and decoder g are allowed too much capacity
  - autoencoder can learn to perform the copying task without learning any useful information about distribution of data
- Autoencoder with a one-dimensional code and a very powerful nonlinear encoder can learn to map $\boldsymbol{x}^{(i)}$ to code i.
  - The decoder can learn to map these integer indices back to the values of specific training examples
- Autoencoder trained for copying task fails to learn anything useful if f/g capacity is too great

Cases when Autoencoder Learning Fails

- Where autoencoders fail to learn anything useful:
  1. Capacity of encoder/decoder f/g is too high
     - Capacity controlled by depth
  2. Hidden code $\boldsymbol{h}$ has dimension equal to input $\boldsymbol{x}$
  3. *Overcomplete* case: where hidden code $\boldsymbol{h}$ has dimension greater than input $\boldsymbol{x}$
     - Even a linear encoder/decoder can learn to copy input to output without learning anything useful about data distribution

What is the Right Autoencoder Design?

- Ideally, choose code size (dimension of h) small and capacity of $f\,/\,g$ based on complexity of distribution modeled
- Alternatively, regularized autoencoders provide the ability to do so
  - Use a loss function that encourages the model to have properties other than copy its input to output

2. Regularized Autoencoders

- Allow the the code to have properties
  - Besides keeping encoder/decoder shallow and code size small
  - Regularized autoencoders have properties other than ability to copy its input to its output
- Other properties include:
  - Sparsity of representation
  - Smallness of the derivative of the representation
  - Robustness to noise
  - Robustness to missing inputs
- Regularized autoencoder can be nonlinear and overcomplete
  - But still learn something useful about data distribution even if model capacity is great enough to learn trivial identity function

Generative Models Viewed as Autoencoders

- Generative models with latent variables and an inference procedure (for computing latent representations given input) can be viewed as a particular form of autoencoder
- Generative modeling approaches which emphasize connection with autoencoders are descendants of Helmholtz machine:
  1. Variational autoencoder
  2. Generative stochastic networks

Sparse Autoencoders

- A sparse autoencoder is an autoencoder whose
  - Training criterion includes a sparsity penalty $\Omega(\boldsymbol{h})$ on the code layer $\boldsymbol{h}$ in addition to the reconstruction error:

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h})$$

  - where $g(\boldsymbol{h})$ is the decoder output and typically we have $\boldsymbol{h} = f(\boldsymbol{x})$
- Sparse encoders are typically used to learn features for another task such as classification
- An autoencoder that has been trained to be sparse must respond to unique statistical features of the dataset rather than simply perform the copying task
  - Thus sparsity penalty can yield a model that has learned useful features as a byproduct

Sparse Encoder doesn't have Bayesian Interpretation

- Penalty term $\Omega(\boldsymbol{h})$ is a regularizer term added to a feedforward network whose
  - Primary task: copy input to output (with *Unsupervised* learning objective)
  - Also perform some supervised task (with *Supervised* learning objective) that depends on the sparse features
- In supervised learning regularization term corresponds to prior probabilities over model parameters
  - Regularized MLE corresponds to maximizing $p(\boldsymbol{\theta}|\boldsymbol{x})$, which is equivalent to maximizing $\log p(\boldsymbol{x}|\boldsymbol{\theta})+\log p(\boldsymbol{\theta})$
    - First term is data log-likelihood and second term is log-prior over parameters
  - Regularizer depends on data and thus is not a prior
    - Instead, regularization terms express a preference over functions

Generative Model view of Sparse Autoencoder

- Rather than thinking of sparsity penalty as a regularizer for copying task, think of sparse autoencoder as approximating ML training of a generative model that has latent variables
- Suppose model has visible/latent variables $\boldsymbol{x}$ and $\boldsymbol{h}$
- Explicit joint distribution is $p_{\text{model}}(\boldsymbol{x},\boldsymbol{h}) = p_{\text{model}}(\boldsymbol{h})\, p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})$
  - where $p_{\text{model}}(\boldsymbol{h})$ is model's prior distribution over latent variables
    - Different from p(θ) being distribution of para
- The log-likelihood can be decomposed as $\boxed{\log p_{\text{model}}(\boldsymbol{x},\boldsymbol{h}) = \log \sum_{\boldsymbol{h}} p_{\text{model}}(\boldsymbol{h},\boldsymbol{x})}$
- Autoencoder approximates the sum with a point estimate for just one highly likely value of $\boldsymbol{h}$, the output of a parametric encoder
  - With a chosen $\boldsymbol{h}$ we are maximizing $\log p_{\text{model}}(\boldsymbol{x},\boldsymbol{h}) = \log p_{\text{model}}(\boldsymbol{h}) + \log p_{\text{model}}(\boldsymbol{x}|\boldsymbol{h})$

Sparsity-inducing Priors

- The $\log p_{\text{model}}(\boldsymbol{h})$ term can be sparsity-inducing. For example the Laplace prior

$$p_{\text{model}}(h_i) = \frac{\lambda}{2} e^{-\lambda |h_i|}$$

  - corresponds to an absolute value sparsity penalty
- Expressing the log-prior as an absolute value penalty

$$\boxed{-\log p_{\text{model}}(\boldsymbol{h}) = \sum_i \left(\lambda \,|\, h_i \,| - \log \frac{\lambda}{2} \right) = \Omega(\boldsymbol{h}) + const} \qquad \text{where} \quad \Omega(\boldsymbol{h}) = \lambda \sum_i h_i$$

  - where the constant term depends only on λ and not on $\boldsymbol{h}$
- We treat λ as a hyperparameter and discard the constant term, since it does not affect parameter learning

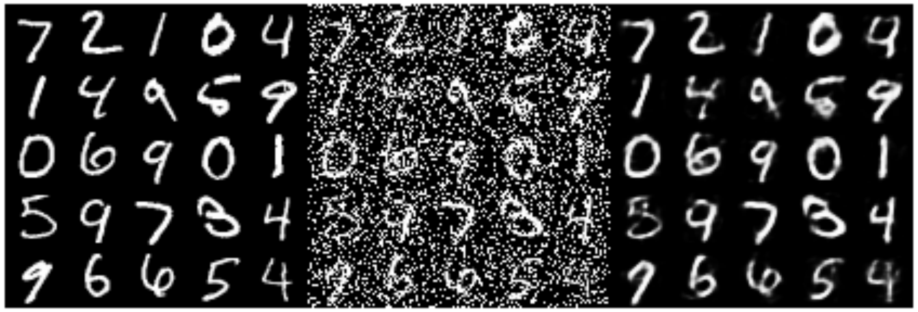
Denoising Autoencoders (DAE)

- Rather than adding a penalty Ω to the cost function, we can obtain an autoencoder that learns something useful
  - By changing the reconstruction error term of the cost function
- Traditional autoencoders minimize $L(\boldsymbol{x}, g(f(\boldsymbol{x})))$
  - where L is a loss function penalizing $g(f(\boldsymbol{x}))$ for being dissimilar from $\boldsymbol{x}$, such as L^2 norm of difference: mean squared error
- A 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
  - The autoencoder must undo this corruption rather than simply copying their input
- Denoising training forces f and g to implicitly learn the structure of $p_{data}(\boldsymbol{x})$
- Another example of how useful properties can emerge as a by-product of minimizing reconstruction error

DAE for MNIST data

Python

- import theano.tensor as T
- from opendeep.models.model import Model
- from opendeep.utils.nnet import get_weights_uniform, get_bias
- from opendeep.utils.noise import salt_and_pepper
- from opendeep.utils.activation import tanh, sigmoid
- from opendeep.utils.cost import binary_crossentropy
- # create our class initialization!
- class DenoisingAutoencoder(Model):
- """
- A denoising autoencoder will corrupt an input (add noise) and try to reconstruct it.
- """
- def __init__(self):
    - # Define some model hyperparameters to work with MNIST images!
    - input_size = 28*28 # dimensions of image
    - hidden_size = 1000 # number of hidden units - generally bigger than input size for DAE
    - # Now, define the symbolic input to the model (Theano)
    - # We use a matrix rather than a vector so that minibatch processing can be done in parallel.
    - x = T.fmatrix("X")
    - self.inputs = [x]
    - # Build the model's parameters - a weight matrix and two bias vectors
    - W = get_weights_uniform(shape=(input_size, hidden_size), name="W")
    - b0 = get_bias(shape=input_size, name="b0")
    - b1 = get_bias(shape=hidden_size, name="b1")
    - self.params = [W, b0, b1]
    - # Perform the computation for a denoising autoencoder!
    - # first, add noise (corrupt) the input
    - corrupted_input = salt_and_pepper(input=x, corruption_level=0.4)
    - # next, compute the hidden layer given the inputs (the encoding function)
    - hiddens = tanh(T.dot(corrupted_input, W) + b1)
    - # finally, create the reconstruction from the hidden layer (we tie the weights with W.T)
    - reconstruction = sigmoid(T.dot(hiddens, W.T) + b0)
    - # the training cost is reconstruction error - with MNIST this is binary cross-entropy
    - self.train_cost = binary_crossentropy(output=reconstruction, target=x)



Unsupervised Denoising Autoencoder
Left: original test images
Center: corrupted noisy images
Right: reconstructed images

Regularizing by Penalizing Derivatives

- Another strategy for regularizing an autoencoder
- Use penalty as in sparse autoencoders

$$L(\boldsymbol{x}, g(f(\boldsymbol{x}))) + \Omega(\boldsymbol{h})$$

- But with a different form of Ω

$$\boxed{\Omega(\boldsymbol{h},\boldsymbol{x}) = \lambda \sum_i \left\| \nabla_x h_i \right\|^2}$$

- Forces the model to learn a function that does not change much when $\boldsymbol{x}$ changes slightly
- Called a Contractive Auto Encoder (CAE)
- This model has theoretical connections to
  - Denoising autoencoders
  - Manifold learning
  - Probabilistic modeling

3. Representational Power, Layer Size and Depth

- Autoencoder often trained with with single layer
- However using deep encoder offers many advantages