Varieties and Applications of Autoencoders

Isabel Restrepo Paul Stey

March 17, 2017

Table of Contents

Autoencoder Basics

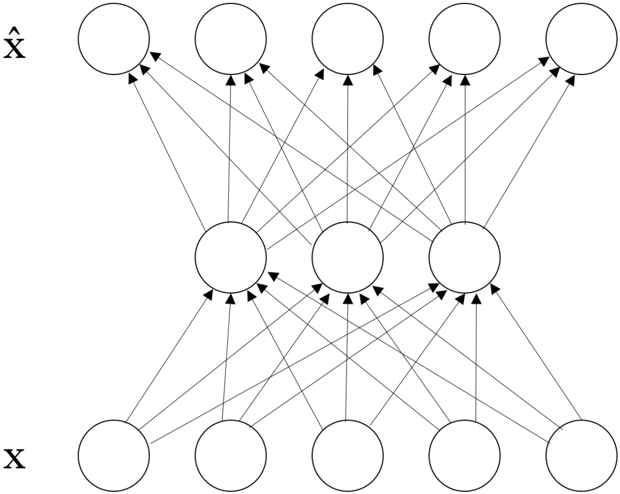
1. Feed-forward neural network
2. Used for unsupervised learning
3. Trained to reproduce input layer in the output layer
4. Training uses gradient descent

Autoencoder Basics cont.

1. Similar to PCA*, but more flexible
2. Autoencoders can accommodate non-linear transformations
    1. Consequence of flexible activation functions
    2. Turns out to be hugely useful

* Actually, with squared error loss and no sigmoid transformation, it is equivalent to PCA (Baldi & Hornik, 1989)

Simple Autoencoder



Simple Autoencoder cont.

From input layer to hidden layer is "encoder":

$$f(x) = \sigma(Wx + b)$$

Hidden layer to the output is the "decoder":

$$g(x) = \sigma(W'x + b')$$

where σ is the activation function (often sigmoid) and W' is often $W^\top$ (known as having "tied" weights).

Simple Autoencoder cont.

1. Input layer $\rightarrow$ hidden layer (fewer units) $\rightarrow$ output layer
2. Hidden layers values can be thought of as lossy compression of input layer
3. Hidden layer with fewer units than input is often called "bottleneck" or under-complete layer

Question

In a sense, autoencoders want to learn an approximation to the identity function.

Why would this be useful?

Answer

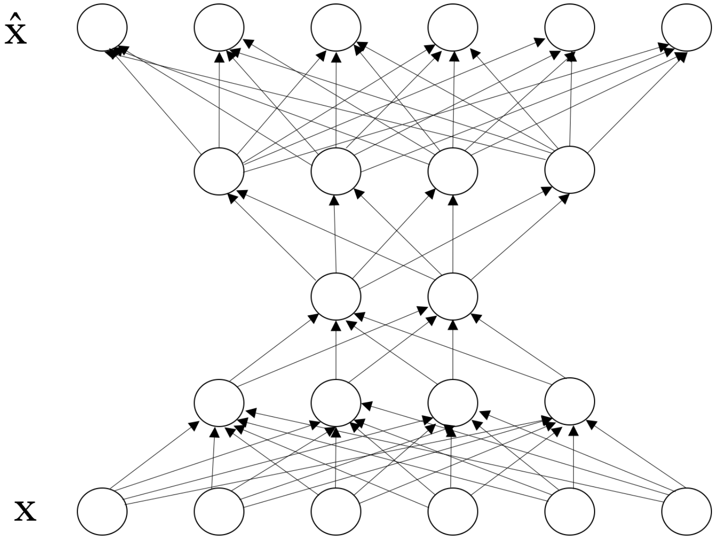
This turns out to have several practical uses.

1. Autoencoders learn a kind of latent representation of the input
2. Can be used for initializing weights and biases prior to fitting neural net
3. Very well suited to dimensionality reduction in NLP tasks

Stacked Autoencoder

1. Sometimes called deep autoencoder
2. Feed-forward neural network
3. Has additional layers
4. Still “unsupervised” in the sense of no labels or real-valued outcome variable
5. Output layer is still $\hat{x}$
6. Trained in stages

Stacked Autoencoder



Stacked Autoencoder cont.

1. Beneficial for data with hierarchical organization
2. Can be difficult to train using back propagation

Sparse Autoencoder

1. Different approach to “compressing” representation
2. Hidden layer has more units than input (i.e., over-complete)
3. Compression is therefore achieved through sparsity

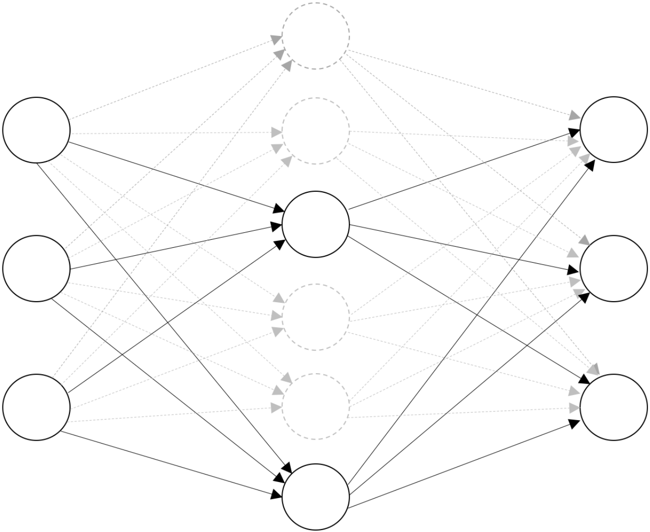
Sparse Autoencoder cont.

1. Several methods of introducing sparsity, we'll discuss 2
   1. k-sparse autoencoders (Makhzani et al., 2013)
   2. Constrained activation

k-sparse Autoencoder

1. Select k highest-activation units
2. Set others to 0
3. Compression is achieved through sparsity of model, not reduced dimensionality

k-sparse Autoencoder



Sparsity Constraint

1. With sigmoidal activation function, $\sigma_j(x) \in [0, 1]$
2. We want to constrain them to be mostly 0
3. Take mean activation

$$\hat{\rho}_j = \frac{1}{n}\sum_{i=1}^{n}\left[\sigma_j\left(x_i\right)\right]$$

such that

$$\hat{\rho}_j = \rho$$

where ρ is a hyperparameter set near 0, commonly 0.05.

Sparsity Constraint cont.

1. Add penalty term to loss function to achieve this sparsity constraint

$$\sum_{j=1}^{m} \rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}_j}$$

2. This is based on Kullback-Leibler Divergence, and often written as

$$\sum_{j=1}^{m} \mathsf{KL}\left(\rho || \hat{\rho}_j\right)$$

Sparsity Constraint cont.

1. Penalty term has the property that KL $(\rho||\hat{\rho}_j) = 0$ when $\hat{\rho}_j = \rho$
2. Otherwise, monotone increasing as $\hat{\rho}_j$ diverges from ρ
3. So our loss function is now

$$L_{\text{sparse}}(x, \hat{x}) = L(x, \hat{x}) + \lambda \sum_{j=1}^{m} \text{KL}\left(\rho||\hat{\rho}_j\right)$$

where $L(x, \hat{x})$ is our unconstrained loss function and λ controls the weight of sparsity penalty.

Denoising Autoencoder cont.

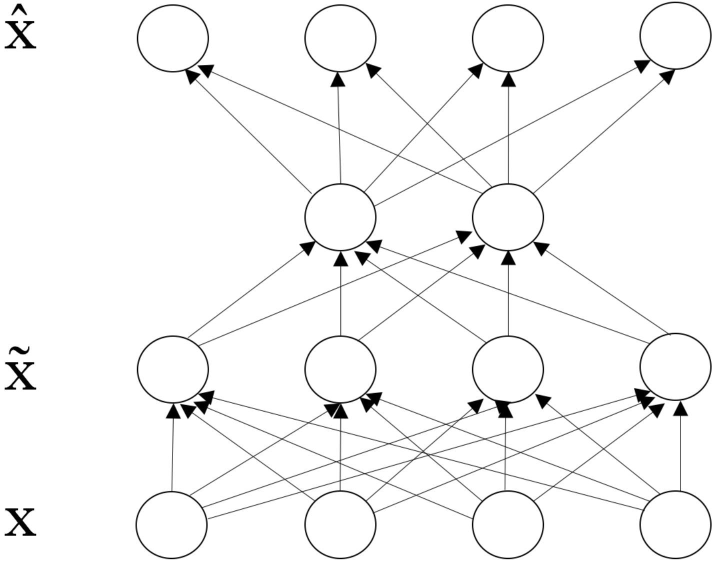
1. Stochastic version of autoencoder
2. Random noise injected into the input layer
   1. *Masking noise*: some fraction of elements* in x set to 0
   2. *Gaussian additive noise*: $\tilde{x}|x \sim \mathcal{N}(x, \sigma^2 I)$
   3. *Salt-and-pepper noise*: fraction of elements* in x set to either min or max possible value (often 0 or 1) according to coin toss

* chosen at random for each example

Denoising Autoencoder

1. Output layer is evaluated against original (uncorrupted) data
2. Goal is slightly different from simple autoencoder
    1. Not attempting to learn identity function
    2. We want robustness of representation
    3. Learned representation is insensitive to perturbations in the input

Denoising Autoencoder



Denoising Autoencoder cont.

1. Training proceeds fairly similarly to standard autoencoder
2. Crucial difference is the loss function is calculated using *uncorrupted* input layer

Extending Denoising Autoencoder

1. Some noise types only corrupt subset of the input's components
   1. Masking noise
   2. Salt-and-pepper noise
2. For these, we can extend denoising by adding emphasis on corrupted components.
3. Use hyperparameters α and β to control emphasis; for squared error loss, this yields

$$L(x, \hat{x}) = \alpha \left(\sum_{i \in \mathcal{I}(\tilde{x})} (x_i - \hat{x}_i) \right) + \beta \left(\sum_{i \notin \mathcal{I}(\tilde{x})} (x_i - \hat{x}_i) \right)$$

where $\mathcal{I}(\tilde{x})$ denotes indexes components of x that were corrupted.

Some Advantages of Autoencoders

1. Compressed representation of features used for initializing weights
   1. Faster learning
   2. Less likely to get stuck in local minima
   3. Better approximation $\rightarrow$ increased accuracy
2. Well suited to dimensionality reduction in NLP problems
3. Allows us to make use of unlabeled data

Disadvantages of Autoencoders

1. Computational burden
2. Limitations of unsupervised learning

Not Discussed

1. Winner-Take-All autoencoders (for sparsity)
2. Convolutional autoencoders (several varieties)
3. Variational autoencoders

References

1 Vincent P., Larochelle, H., Bengio Y., Manzagol P.A. (2008) *Extracting and Composing Robust Features with Denoising Autoencoders*.

2 Vincent P., Larochelle, H., LaJoie I., Bengio Y., Manzagol P.A. (2010) *Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion*

3 Makhzani A., Frey B. (2015) *Winner-Take-All Autoencoders*

4 Ng A. *Sparse Autoencoders* CS294A Lecture notes