

AutoEncoder

Intro Notes

What is the purpose of AutoEncoders? (What do Autoencoders do?)

Autoencoders are neural networks that are trained to attempt to copy their input to their output in an imperfect manner

What are the parts of an AutoEncoder?

1. Encoder Function ($h=f(x)$)
2. Decoder Function that produces reconstruction, $r=g(h)$

Why do AutoEncoders copy in an imperfect manner?

1. Perfect copying isn't especially useful or interesting
↳
2. The model is forced to prioritize aspects of the data to be copied, which inadvertently causes the model to learn important aspects of the inputted data
3. Finding a low (relatively) - dimension representation of input data

Other General Notes: Autoencoders are similar to feedforward nets and \therefore can be trained in the same ways (eg. backprop, adam, etc)

Sparse Autoencoder Notes

Training...

- Involves a sparsity penalty, $\Omega(h)$, on the network's internal representation, h .
- The reconstruction error is as follows (put this eq. on slide)
$$\mathcal{L}(x, g(f(x))) + \Omega(h)$$

Uses...

- Feature Extraction
- Sparse Representation of Input Data
 - ↳ What's the point of having a sparse representation?
 - Sparse AEs respond to statistical features of the dataset, which allows one to learn more important info about the input

What is the point of Ω function?

Let $\hat{p}_j = \frac{1}{m} \sum_{i=1}^m [a_j^{(2)} x^{(i)}]$ be the avg activation of hidden unit j over the whole dataset

Goal: Enforce the following constraint, $\hat{p}_j = p$, where $p =$ "a sparsity parameter" that is close to zero. In order to satisfy this constraint, most hidden activations must be near zero.

To achieve the above goal, we decide to penalize deviations of \hat{p}_j from p 's value by adding a penalty term to the optimization objective

$$\left[\sum_{j=1}^S p \log \frac{p}{\hat{p}_j} + (1-p) \log \frac{1-p}{1-\hat{p}_j} \right] \approx \Omega(h)$$

Further Notes on the Ω Function

The term introduced on the last page is commonly referred to as the KL divergence function, $\sum_{j=1}^S KL(p \parallel \hat{p}_j)$

$$\text{where } KL(p \parallel \hat{p}_j) = p \log \frac{p}{\hat{p}_j} + (1-p) \log \frac{1-p}{1-\hat{p}_j}$$

(KL divergence b/t a Bernoulli RV w/ mean p and a Bernoulli RV w/ mean \hat{p}_j)

So, in this context (forcing sparsity on an AE), the KL divergence function acts as the Ω function

Undercomplete Autoencoders

What is an undercomplete Autoencoder?

- An undercomplete autoencoder has a hidden layer, h , with fewer dimensions than ^{that of} x , the input

What is the learning function?

$L(x, g(f(x)))$ where L is a loss function penalizing $g(f(x))$ for being dissimilar from x (ex. MSE)

Abilities...

- If decoder is linear and L is MSE, an undercomplete AE learns to span the same subspace as a PCA
- If ~~decoder~~ ^{encoder} is nonlinear and ~~it is MSE, along with encoder,~~ then ~~the~~ nonlinear decoder, g , can learn a more powerful nonlinear generalization of PCA

Potential Problems...

- Capacity becoming too great w/out appropriate constraints could cause a lack of effective learning in the hidden layer
- Powerful nonlinear encoder, if not used properly, could also cause a lack of learning

Autoencoder Example

- Say we have a 10×10 pixelated image where ~~each pixel~~ ~~is an input to our autoencoder~~ and each input, x , is the array of pixel color values from the image
 - So, $n=100$ and let's say that there are $8_2=50$ hidden units in layer L_2
 - $y \in \mathbb{R}^{100}$ and since we have only 50 hidden units, the network must put together a more condensed version of the ~~inputted~~ inputted data. What this means for us is that the autoencoder now has to try and recreate the original image using only the vector of hidden unit activations ($a \in \mathbb{R}^{50}$)
 - In this way, the autoencoder is able to learn key low-dimensional representations of inputted data
- where a_j is the activation of hidden unit j of the autoencoder

Autoencoder Architecture Notes

- Since both the encoder and decoder are feedforward networks, they can have stacked layers, and benefit from the addition of more layers
- But, it is fairly common for the encoder and decoder to be single layers

Universal Approximator Theorem...

Arbitrary constraints can be enforced
w/ at least 1 additional hidden layer inside encoder
can approximate any mapping from input to code layer,
~~code~~ given enough hidden units in code layer

Deep AEs in practice...

- ~~Red~~ Exponentially reduced computational cost of representing some functions
- Yield much better compression

Stochastic Encoder/Decoders Notes

- Approximate data distribution of an input x
 - A probabilistic encoder $q_{\phi}(x|z)$ is used to produce a gaussian distribution in the feature representation space of the input
 - A probabilistic decoder $p_{\theta}(z|x)$ is used to produce a probabilistic distribution over the input space

Overall Goal:

- Picking a family of distributions over the latent (hidden layer) ^{code} variables w/ variational parameters $q_{\phi}(z)$ and estimate the parameters for the resulting family st $\rightarrow q_{\phi}$ (some optimal value)