Autoencoders

Advanced Institute for Artificial Intelligence – AI2

https://advancedinstitute.ai

Introduction

What are Autoencoders?

Definitions

- "Unsupervised" learning algorithm that applies backpropagation, setting the target values to be equal to the inputs:
  - uses $y^{(i)} = x^{(i)}$
- Internally, it has a hidden layer that describes a code used to represent the input;
- Consists of two parts:
  - Encoder function $h = f(x)$, and
  - Decoder function that produces a **reconstruction** $r = g(h)$;
- Designed to be **unable to learn to copy perfectly**;

Question

Why simply learning to set $g(f(x)) = x$ everywhere is not especially useful?

Definitions

- Special cases of feedforward networks and may be trained with all the same techniques (minibatch gradient descent, back-propagation, etc.);
- Usually restricted to copy only approximately, i.e., producing data **that only resembles the training data**.
- The model is forced to prioritize only a few aspects of the input;
  - Often learns useful properties of the data, e.g., relevant features;
- Traditionally used for **dimensionality reduction** or **feature learning**;

Example

Suppose the inputs x are the pixel intensity values from a **10×10 image** (100 pixels) - n=100, and there are $s_2 = 50$ hidden units in layer L_2.

From the definition of Autoencoders we have $y \in \mathcal{R}^{100}$. Since there are only 50 hidden units, the network is forced to learn a "compressed" representation of the input.

Given only the vector of hidden unit activations $a^{(2)} \in \mathcal{R}^{50}$, it **must try to "reconstruct"** the 100-pixel input x.

Definitions

- If there is some underlying structure in the data, e.g., some of the input features are correlated, then this algorithm **will be able to discover some of those correlations**.
- This simple form of autoencoder most likely will learn a low-dimensional representation very similar to PCAs.
- Can be thought of as **data compression algorithms**;
- Compression and decompression functions are:
  1. data-specific,
  2. lossy, and
  3. learned automatically from examples rather than engineered by a human;

Question

1. Why Autoencoders wouldn't make great **general-purpose** compression algorithms?
2. Why do they **need** to be lossy?
3. An autoencoder trained on pictures of faces would do a good job on compressing pictures of trees?

What are they good for?

- Data compression?
  - Almost impossible to beat standard algorithms, such as JPEG, MP3, etc.;
  - You can **improve the performance by restricting the type of data** it uses;
    - **Loss of generalization capability!**
  - Generally impractical for real-world data compression problems:
    - Can only be used on data that is similar to what they were trained on.
- Dimensionality reduction:
  - If the decoder is linear and the cost function is the Mean Square Error, an Autoencoder learns to span the same subspace as the PCA;
- Data denoising
  - the data is partially corrupted by noises;
  - the model is trained to predict the original, uncorrupted data point as its output;

Undercomplete Autoencoders

Undercomplete Autoencoders

- Autoencoder whose code dimension is less than the input dimension;
- Forces the autoencoder to capture the most relevant features of the training data;
    - Also known as bottlenecks;
- Minimize the Loss function, where f is the function learned by the encoder and g is the function learned by the decoder by tweaking the parameters θ and ϕ:

$$L_{AE}(\theta, \phi) = \frac{1}{n} \sum_{i=1}^{n} (x^{(i)} - f_{\theta}(g_{\phi}(x^{(i)})))^2$$

- Nonlinear encoder functions f and nonlinear decoder functions g can learn a more powerful nonlinear generalization of PCA;

Implementation

- General Architecture:

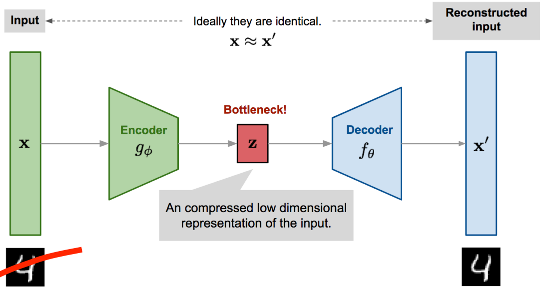

Figure: General architecture of an undercomplete autoencoder

Image from Autoencoder architecture by Lilian Weng

Implementation

- Dimensionality Reduction for a 3D dataset

```
encoding_dim = 2
input_layer = keras.Input(shape=(3,))
encoded = layers.Dense(encoding_dim, activation="sigmoid")(input_layer)
decoded = layers.Dense(3, activation="sigmoid")(encoded)
autoencoder = keras.Model(input_layer, decoded)
autoencoder.compile(loss="mse", optimizer="SGD")
```

Implementations

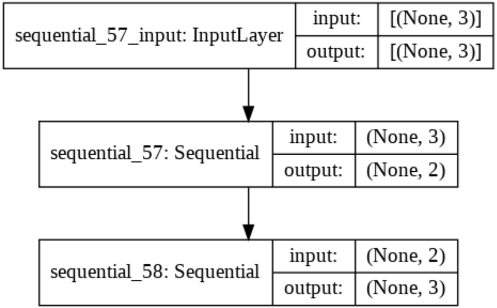

Figure: Autoencoder Architecture

Undercomplete Autoencoders

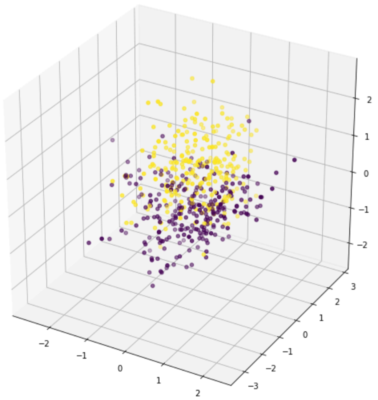

Figure: 3D data before encoding

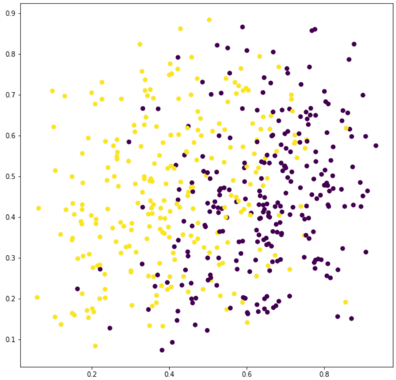

Figure: 2D data after encoding

Undercomplete Autoencoders

Jupyter Notebook with example and exercises.

What they are not good at?

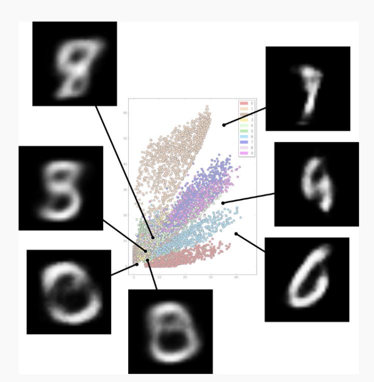

Figure: Latent Space representation for the MNIST dataset

What they are not good at?

- Some of the biggest challenges regarding the latent space are:
  - **Gaps in the latent space**: we do not know what data points in these spaces may look like
  - **Separability in the latent space**: there are also regions where the labeled is randomly interspersed
  - **Discrete latent space**: we do not have a statistical model that has been trained for arbitrary input

Under/over complete Autoencoders

Limitations

- Unfortunately, undercomplete autoencoders **fail to learn anything useful** if the encoder and decoder **are given too much capacity**
- Also occurs if the hidden code has the same dimension as the input;
- The same happens in the overcomplete case, where the hidden code has dimension greater than the input;
- Even a linear encoder and decoder can **learn to copy the input** to the output
  - **Nothing useful is learned about the data distribution**

Regularized Autoencoders

Regularized Autoencoders

- Use a loss function that encourages the model to have other properties, e.g., sparsity of the representation, and robustness to noise or to missing inputs;
- Can be nonlinear and overcomplete but still learn something useful about the data distribution;
- The two of the most common Regularized Autoencoders are:
  - **Sparse Autoencoders**: sparsity penalty added to his original loss function;
  - **Denoising Autoencoders**: adding noise (Gaussian for example) to the inputs forces our model to learn important features;

Sparse Autoencoders

- Is simply an autoencoder whose training criterion involves a **sparsity penalty**;
- Presents a larger latent dimension than the input or output dimensions;
- Typically used to **learn features for another task**, such as classification;
- Think of the penalty simply as a **regularizer term**;
- We would like to constrain the neurons to be **inactive most of the time**;
- Reduce the propensity for the network to overfit;
- It can **no longer copy the input through certain nodes**:
  - in each run, **those nodes may not be the active**

Sparse Autoencoders

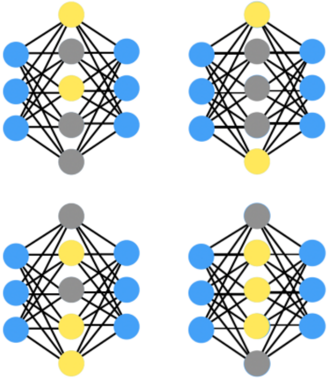

Image by Shreya Chaudhary

Sparse Autoencoders - Implementation

In Keras, this can be done by adding an `activity_regularizer` to our `Dense` layer

```
encoded = layers.Dense(encoding_dim, activation='relu',
                activity_regularizer=regularizers.l1(10e-5))(input_img)
```

With the added regularization the model is less likely to overfit and **can be trained longer**;

- Example Notebook for the MNIST dataset;

Denoising Autoencoders

- The input is partially corrupted by adding noises to or masking some values of the input vector in a stochastic manner;
- the model is trained to recover the original input (note: not the corrupt one);

$$\tilde{\mathbf{x}}^{(i)} \sim \mathcal{M}_{\mathcal{D}}(\tilde{\mathbf{x}}^{(i)}|\mathbf{x}^{(i)})$$

$$L_{\mathsf{DAE}}(\theta, \phi) = \frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}^{(i)} - f_\theta(g_\phi(\tilde{\mathbf{x}}^{(i)})))^2$$

where $\mathcal{M}_{\mathcal{D}}$ defines the mapping from the true data samples to the noisy or corrupted ones.

Denoising Autoencoders

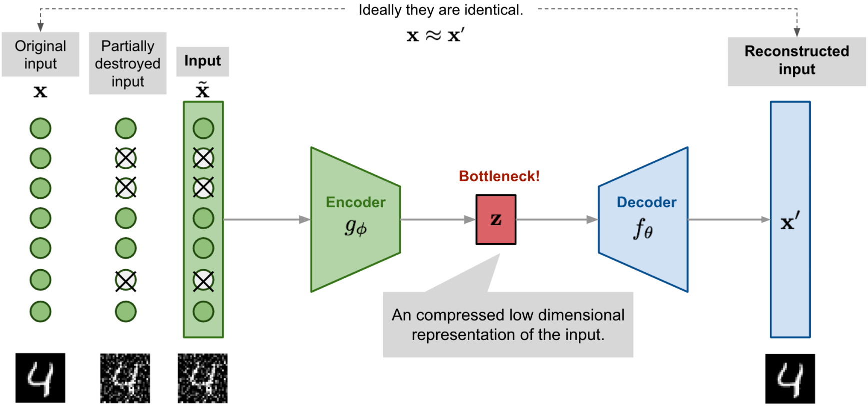

Image by Lilian Weng

Denoising Autoencoders

- Design motivated by the fact that humans can easily recognize an object even the view partially occluded;
- To "repair" the input, the DAE has to discover the relationship between dimensions of input in order to infer missing pieces;
- On images, the model is likely to depend on evidence gathered from a combination of many input dimensions to recover the denoised version;
    - This builds up a good foundation for learning robust latent representation;
- In the original DAE paper, a fixed proportion of input dimensions are selected at random and their values are forced to 0 (same as dropout?;

Regularized Autoencoders

Denoising Autoencoders - Implementation

In the case of the MNIST dataset:

```
noise_factor = 0.5
x_train_noisy = x_train + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_train.shape)
x_test_noisy = x_test + noise_factor * np.random.normal(loc=0.0, scale=1.0, size=x_test.shape)
```

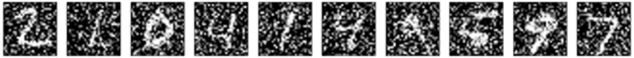

Figure: MNIST digits after adding Noise

Jupyter Notebook with the example implementation of Denoising Autoencoder for the MNIST.