Autoencoders

Andrew H. Fagg
Autoencoders
Autoencoders

● Unsupervised learning: there is no separate “desired output” from the network
  ○ Data can be a lot easier to come by
● Central layer is the compressed representation of the input
  ○ Must preserve the information content of the input, but with fewer dimensions
  ○ “Latent representation”
Latent Representations

Can be used as:

- Inputs to other networks
  - Transfer learning: further training with a labeled data set
  - Tend to have less noise than the original input, so less prone to overfitting
- Visualization of the high-dimensional input
  - Often need further compression to do this: PCA, ISOmap, tSNE
Latent Representations

Can be grown incrementally:

● Start with training a shallow network
● Keep the encoder, but then add:
  ○ A more compressed encoder
  ○ A full decoder
● Train again
● Repeat
Convolutional Autoencoders

- Input / output are images
- Encoder: reduce the spatial resolution at each step
- Decoder: increase resolution
Convolutional Autoencoders

medium.com/@birla.deepak26
Convolutional Autoencoders

Encoder:

- Spatial resolution generally reduces at each step (Convolution + striding)
- Number of channels increases
- So: trading spatial resolution for resolution in the channels
- But: \((r \times c) / \text{ch}\) still will generally drop with each step
Convolutional Autoencoders

Decoder: increase resolution at some steps

- Conv 2D Transpose: kernel maps one pixel in the input to $k \times k$ pixels in the output
- Upsample: Copy one pixel in the input to $k \times k$ pixels in the output

Because the former can lead to strange artifacts, the latter is preferred practice today
Convolutional Autoencoders: Practice

Can be hard to end up with the same dimensions on the input and output sides of the autoencoder

- Keep kernel size and stride the same
- Only choose kernel sizes to be integer factors of the image size
- Middle-most layer: can bring to a 1x1 image
  - Vector summarizes the image a non-spatial manner
  - Latent representation of the input
Autoencoders:
Dealing with Training Set Size

● When training set size is small, we run the risk of capturing the noise in the image, as well as the real structure

● One approach: data augmentation
  ○ Augment training set with additional training samples derived from the original training set
Data Augmentation

A cat is still a cat if:

- Shifted laterally or vertically
- Rotated
- Scaled
- :

Keras ImageDataGenerator class will augment an image set on the fly
Data Augmentation and Autoencoders

- Want our autoencoder to capture the ‘real’ aspects of the image and not the noise
- Denoising autoencoder:
  - Select training image
  - Add pixel-level noise (typically Gaussian-distributed)
  - Input: noisy image
  - Desired output: original image
Developing Sparse Representations

Goal: want very different input images to have very different latent representations (best case: vectors are orthogonal)

- Can add a regularization term that punishes similar representations
- Activity regularization
- Kullback-Leibler divergence
  - Measure of the difference between two distributions
KL math

KL vs MSE
Variational Autoencoder

- Encoder output:
  - Mean and standard deviation in the latent space
- Latent representation: sampled from this Gaussian distribution
- Decoder output:
  - Desire is to recover the original image
Variational Autoencoder

- Reconstruction loss: difference between input and output images
- But: this alone will generally force the standard deviation to zero
- Add a regularization term:
  - Expected distribution in latent space is $\mathcal{N}(0,1)$
  - Measure KL divergence between $\mathcal{N}(0,1)$ and $\mathcal{N}(\mu, \sigma)$
VAE math
Variational Autoencoder

Regularization implications

- The training samples in the latent space must be $N(0,1)$
- Nice property: the weighted average between any two samples is still covered by the distribution
  - Can often result in a decoded mean being meaningful
- But: strange that samples from very different classes should still fall as one $N(0,1)$
  - Really expect non-overlapping clusters
Image to Image Translation

- If we have the labeled data set, we don’t have to reconstruct the same image
- Instead, could reconstruct different images
  - Remove noise
  - Make some semantic change to the image (e.g., changing seasons)
  - Label pixels by their semantic role in the image
Forms of Segmentation

towardsdatascience.com/semantic-segmentation-popular-architectures-dff0a75f39d0
Semantic Segmentation

- What kind of an object are we looking at?
- What type of a role does the object play in the image?

Both: what is the class of each pixel?

Challenge: need images labeled at the pixel level
Encoder/Decoder for Segmentation
Encoder/Decoder for Segmentation
U-Net Architecture

- **U**: compressed representation
  - More abstraction
- **Skip connections**
  - Less abstraction
  - Shallower pathway for learning
Homework 7

Chesapeake Watershed Land Cover data set

- https://www.radiant.earth/mlhub/
- “Patches” data set
- 1 pixel =~ 1 foot^2
- Data for each pixel: various imaging sensors + label
Chesapeake Watershed Land Cover

- Images: (R, G, B, NIR) x 2
- Leaf on: Landsat 8 surface reflectance (9 bands)
- Leaf off: Landsat 8 surface reflectance (9 bands)
Data Details

- **Input:** 256 x 256 x N
  - N = 24 (?)
- **Output:** 256 x 256 x K
  - 1 = water
  - 2 = tree canopy / forest
  - 3 = low vegetation / field
  - 4 = barren land
  - 5 = impervious (other)
  - 6 = impervious (road)
  - 15 = no data
Data Details

● We are only focused on the Pennsylvania portion of the data set
● 50,000 examples for training
  ○ Compressed images: ~20GB
● Will provide:
  ○ Data on OSCER
  ○ Data loader
  ○ Probably a generator that dynamically loads from the disk
Network Architecture

- Input: images
- Output: probability distribution over classes (for each pixel!)
- In between:
  - Start simple
  - Grow the network, as needed