Machine Learning

Autoencoders

Gabriele Russo Russo   Francesco Lo Presti

Laurea Magistrale in Ingegneria Informatica – A.Y. 2023/24

TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

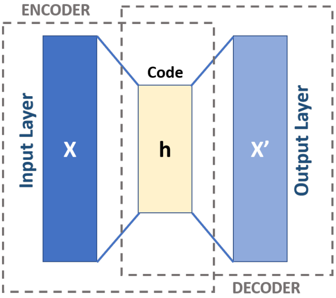
Dipartimento di Ingegneria Civile e Ingegneria Informatica

Autoencoder

- An autoencoder is a neural network trained to attempt to copy its input to its output
- Unsupervised learning model
- Internally, it has a hidden layer $\boldsymbol{h}$ that computes a latent representation (or, coding) of the input data

Two main blocks:

- Encoder: computes $\boldsymbol{h} = f(\boldsymbol{x})$
- Decoder: reconstructs $\boldsymbol{x}' = g(\boldsymbol{h})$



Autoencoder: Why?

- Copying input to output looks like a trivial task (and not particularly useful)
- Autoencoders are usually constrained in various ways, making their task more challenging and useful
  - e.g., constraints on the latent representation

Autoencoder: Example Applications

- Dimensionality reduction: coding forced to have much lower dimensionality than input
  - we care most about the “compressed” latent representation
  - encoder and decoder may be executed on different hosts (e.g., compressing satellite imagery before transmission)
- Noise reduction: autoencoder trained to reconstruct original input x, given a noisy version of x
  - we care most about the reconstruction output
- Anomaly detection: we observe the reconstruction error; a large error may correspond to an anomalous input
  - we care most about the reconstruction error

Simple Autoencoder

- Encoder and decoder may be "simple" feedforward NNs
  - In many cases, they have a single hidden layer
- An additional constraint: decoder's output layer must have the same number of units as encoder's input layer
- How to train the autoencoder? Same algorithms we have used so far ...
- Which loss function? We need a reconstruction loss, to evaluate how well the reconstructed input matches the original input, e.g.:

$$\mathcal{L}(x, x') = ||x - x'||_2^2$$

Example

- Let's train a simple autoencoder on the MNIST (and Fashion MNIST) datasets
- Recall: each image is a 28x28 matrix (or, a 784-element vector) of grayscale pixels
- In this case, we could also exploit binary cross-entropy as the reconstruction loss
  - Think of reconstructing a pixels as computing the probability of the pixel being black
  - Then, compare the output probability to the ground-truth value given by the input image

autoencoder_mnist.ipynb

Deep Autoencoders

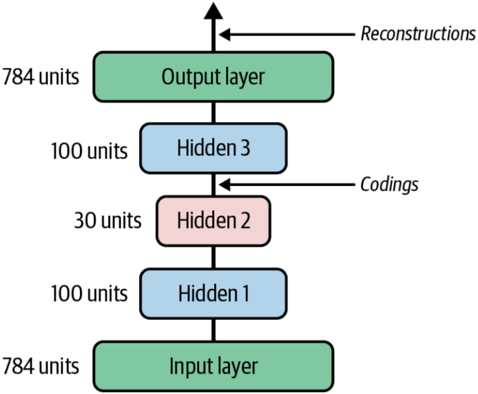
- As usual, we can stack multiple hidden layers to build a stacked (or, deep) autoencoder
- Depth allows the autoencoder to learn more complex codings, and can exponentially decrease the amount of needed training data
- Experimentally, deep autoencoders yield better compression
- But, … training can become more challenging (as usual)
- …a too powerful autoencoder might become useless (see next example)

Example: an Useless Autoencoder

- Consider an extreme case, where an autoencoder has many hidden layers and units
- It might be able to learn a “perfect” representation of the training data
  - Any input encoded just by an integer
  - $x_0 \to 0, x_1 \to 1, \ldots$
  - Decoder learns the inverse mapping
- Clearly, this model would be perfectly useless on new data

Deep Autoencoders (2)

While not necessary, deep autoencoders usually have a symmetrical architecture with a central hidden layer (coding layer)



Tying Weights

- To reduce the number of parameters in a symmetrical deep autoencoder, we can tie the weights of the decoder layers to the weights of the encoder layers
- "Mirrored" layers in the encoder and the decoder share the same weights (just transposed)

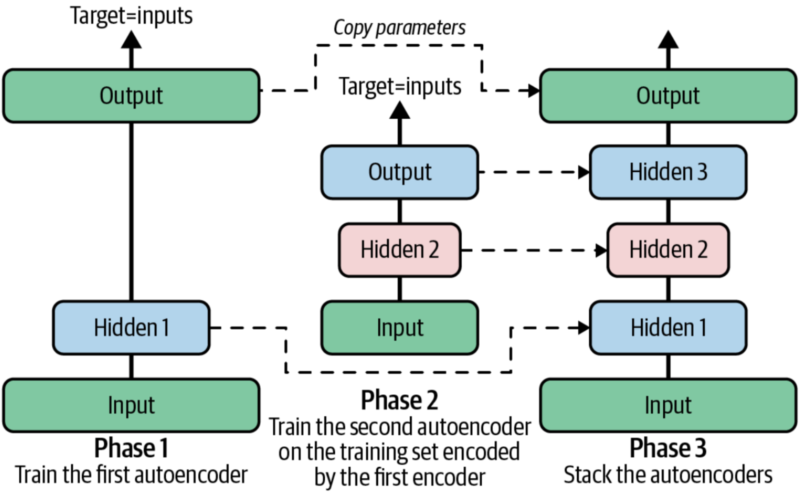
$$w_{N-\ell+1} = w_\ell^T$$

- Tying weights can make the training faster and also reduce overfitting

Training Stacked Autoencoders

- Deep autoencoders can be trained as any deep NN
- Alternatively, we can train one shallow autoencoder at a time, as each needs only learn to reconstruct its own inputs
  - training each shallow AE is simpler
  - final performance might be lower
- Start with the outermost layers and train the first AE; encodings of this AE used as inputs to the next inner AE, …

Training Stacked Autoencoders (2)

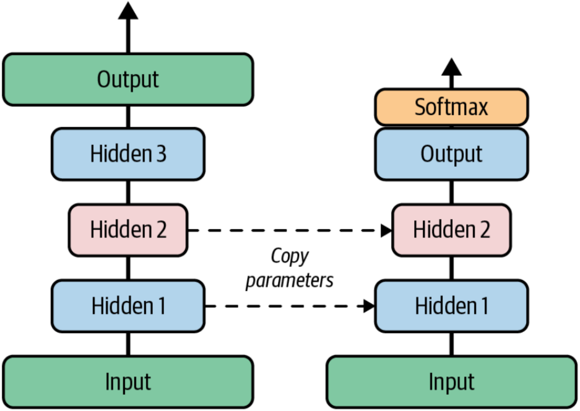


Unsupervised Pretraining

- Imagine a supervised learning task where you have a lot of data, but only few instances are labeled
  - This is a common situation!
  - It is quite easy to gather unlabeled data (e.g., downloading tons of pictures from the web); getting labels is the hard part!
- Training a supervised model with very few instances can be difficult and likely leads to overfitting

- Idea: training an autoencoder on the whole data set
- We can reuse the lower layers, which have learned a representation of the data (i.e., features), to build the final model

Unsupervised Pretraining (2)



- To avoid overfitting, we may freeze the lower layers when training the final model

Convolutional Autoencoders

- Autoencoders are not restricted to fully connected layers
- For instance, when dealing with images, it is natural to exploit convolutional layers
- Encoder comprises traditional convolution and pooling
- However, decoder needs to reconstruct the original image and uses transpose convolutional layers
  - we mentioned them when talking about semantic segmentation
  - think of them as convolution with fractional stride < 1

autoencoder_mnist.ipynb

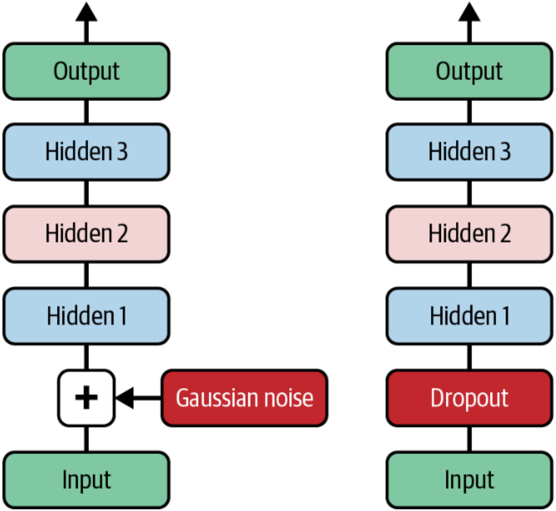
Beyond Undercomplete Autoencoders

- So far, we considered undercomplete autoencoders, where the coding dimensionality is constrained to be smaller than input dimensionality
  - The AE cannot just forward its input to the output
  - Must learn relevant features
- There are situations where we use other kinds of constraints, allowing the coding layer to be as large as the input, or even larger (overcomplete autoencoder)

Denoising Autoencoders (DAE)

- We add noise to the inputs, with the AE trying to reconstruct the original, noise-free input
- Noise can correspond to pure Gaussian noise added to the inputs
- Alternatively, noise can correspond to randomly switched-off inputs (i.e., dropout)

Denoising Autoencoders (2)



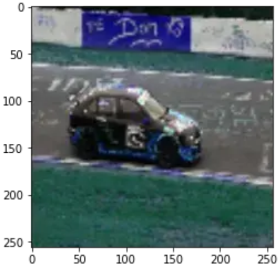
autoencoder_denoising.ipynb

Sparse Autoencoders

- Sparse AE provide an alternative tool for feature extraction
- Rather than using a small number of units in the coding layer, we introduce a sparsity penalty in the training objective
- We aim to reduce the number of active (i.e., non-zero) neurons in the coding layer
- Simple approach (better alternatives exist):
  - Sigmoid activation function in the coding layer
  - L1-regularization applied to layer's activations (not to the weights!)

Image Super-resolution (ISR)

- Image super-resolution: given a low-resolution input image x, reconstruct the high-resolution image x'
- Various approaches to the problem, which is actively studied
  - e.g., CNN-based solution from 2014: `https://arxiv.org/pdf/1501.00092.pdf`



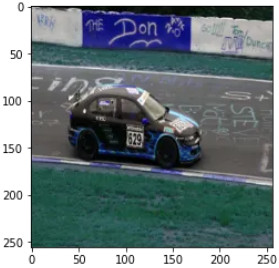


Image Super-resolution (ISR) (2)

- A convolutional AE can be used for ISR, similar to the denoising case
- Suppose you want to train an AE to generate *HxH* images from *LxL* images
- Train the AE with a collection of high-resolution *HxH* images
- Apply a "noise" to the image, by resizing it to *LxL* and then upscaling it back to *HxH*
- Use the AE to reconstruct a noise-free version that matches the original image

`https://medium.com/analytics-vidhya/super-resolution-using-autoencoders-and-tf2-0-505215c1674`

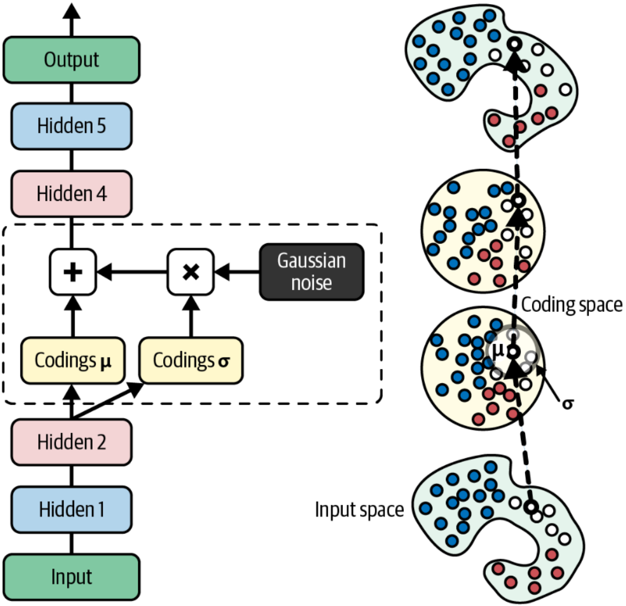
Variational Autoencoders (VAE)

- Quite different from all the autoencoders discussed so far
- Probabilistic AE: outputs are partly determined by chance, even after training (as opposed to denoising AE, which use randomness only during training)
- Generative AE: can generate new instances that look like they were sampled from the training set

Variational Autoencoders (2)

- Same structure as usual: encoder + decoder
- Encoder does not directly produce a coding for an input, but rather the parameters of a distribution
  - e.g., assuming Gaussian distribution,
    encoder produces μ and σ
- Actual coding sampled from the resulting distribution
  - random sampling would make backpropagation difficult
  - in practice, we compute $\boldsymbol{z} = \mu + \sigma \cdot \epsilon$, where ϵ is normally distributed noise
- Decoder works as usual, trying to produce an output that resembles the input instance

Variational Autoencoders (3)



Variational Autoencoders (4)

- The cost function used for training comprises two terms:
  - Reconstruction loss (as usual)
  - Latent loss: forces the VAE to produce codings that look like they have been sampled from the desired distribution (e.g., Gaussian)
- Example with MNIST: `https://www.tensorflow.org/tutorials/generative/cvae?hl=en`
- VAE enjoyed some popularity, but nowadays better (and more complex) generative models exist (e.g., GANs)

Generative Adversarial Networks (GAN)

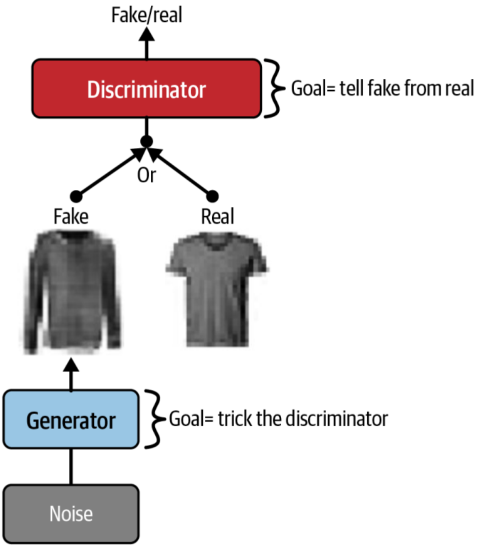
Note: GANs are out of the scope of this lecture on Autoencoders; since we mentioned them, it is worth spending a few words

- GANs were proposed in 2014 by Ian Goodfellow et al.
  - `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`
- Immediate excitement in the community, although it took a few years to overcome training difficulties
- "The most interesting idea in 10 years of ML" (Y. Le Cun, 2016)
- The core idea is simple: make two NNs compete against each other in the hope that this competition will push them to excel

GANs (2)

- Generator network: takes random noise in input (typically Gaussian) and outputs some data (e.g., an image).
  - If you think of the random inputs as the coding of the image to be generated, the generator offers the same functionality as a decoder in a VAE
- Discriminator network: takes an image in input and classifies it either as real (i.e., coming from the training set) or fake (i.e., generated)

GANs (3)



Fake/real
Discriminator
Goal= tell fake from real
Or
Fake
Real
Generator
Goal= trick the discriminator
Noise

GANs: Training

- Each training iteration is divided into two phases
- Phase 1: train the discriminator
  - we use a batch of real images sampled from the training set along with an equal number of fake images produced by the generator
  - binary cross-entropy loss (simple classification task)
- Phase 2: train the generator
  - In the forward pass, the generator produces a batch of images
  - We label all of them as “real” and pass them to the discriminator
  - If the discriminator detects them as fake, we have non-zero loss and use the gradients to train the generator only

Example: StyleGAN (2018)

https://github.com/NVlabs/stylegan

GANs and beyond

- The generator never actually sees any real images, yet it gradually learns to produce convincing fake images!
  - All it gets is the gradients flowing back through the discriminator
- GANs were the state-of-the-art solution for image generation until 2020
- Since 2020, diffusion models have enabled rapid advancements in generative AI for images (e.g., DALL-E, Stable Diffusion)
  - `https://stability.ai/stable-image` is open-source