Deep Neural Networks

Máster Universitario en Ciencia de Datos - Métodos Avanzados en Aprendizaje Automático

Carlos María Alaíz Gudín

Escuela Politécnica Superior
Universidad Autónoma de Madrid

Academic Year 2020/21

UAM
Universidad Autónoma
de Madrid

Contents

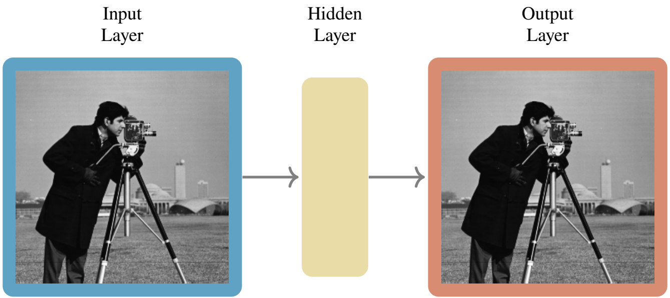
Autoencoders

Autoencoders (I)

Definition (Autoencoder)

An **AutoEncoder** (AE) is a type of NN that learns to produce at the output the exact information received as input.

- The input and output layers should have the same number of units.
- The key is the hidden layer, where the network stores an abstract inner representation of the information.



Autoencoders (II)

- The learning of the AEs is **unsupervised**.

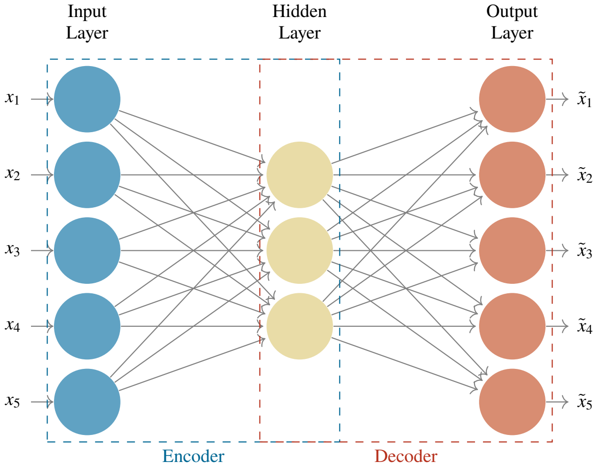
- Given a set of unlabelled data $\mathcal{D} = \{(\mathbf{x}_i)\}_{i=1}^N$, the AE tries to model the identity $f(\mathbf{x}_i) = \mathbf{x}_i$.
- How will the AE solve this problem?

  Desired Behaviour: The AE learns to encode (and compress) the data in the hidden layers.

  Trivial Behaviour: The AE copies the input $\mathbf{x}_i$ layer by layer.

- To prevent the trivial behaviour, the AE can be forced to compress the data.
- The hidden layer should be (much) smaller than the input dimension.
- The AE can be divided in two subnetworks.
  1. The encoder, where the feature extraction takes place.
  2. The decoder, where the feature extraction is reverted.

Autoencoders (III)



Notebook

Simple Autoencoders

Sparse Autoencoders (I)

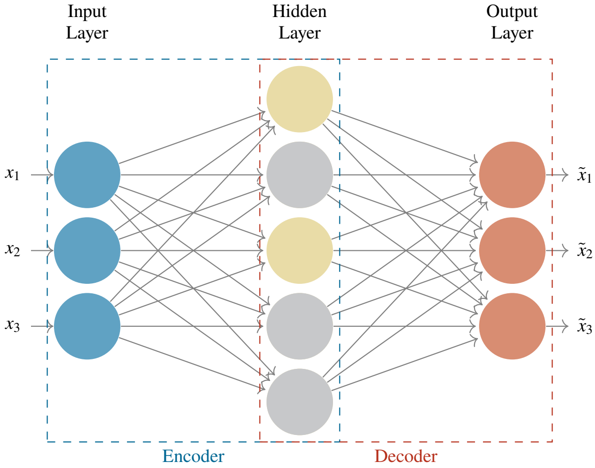
- Can something similar be done if the hidden layer is larger than the input dimension?

- The **Sparse AEs** are forced to provide a sparse codification of the data.
  - A sparse-inducing regularizer is added.
- Only a subset of the hidden units are active at the same time.

- As before, the sparse AE can be divided in two subnetworks.
  1. The sparse encoder, where the data is transformed using a sparse code.
  2. The sparse decoder, where the sparse code is transformed back into the original data.

- To guarantee the generalization, some noise can be added to the input, while keeping the target clean.
  - The number of patterns can be synthetically increased.
  - The AE has to learn to denoise the input through the mapping $f(\mathbf{x}_i + \epsilon_i) = \mathbf{x}_i$.

Sparse Autoencoders (II)



Notebook

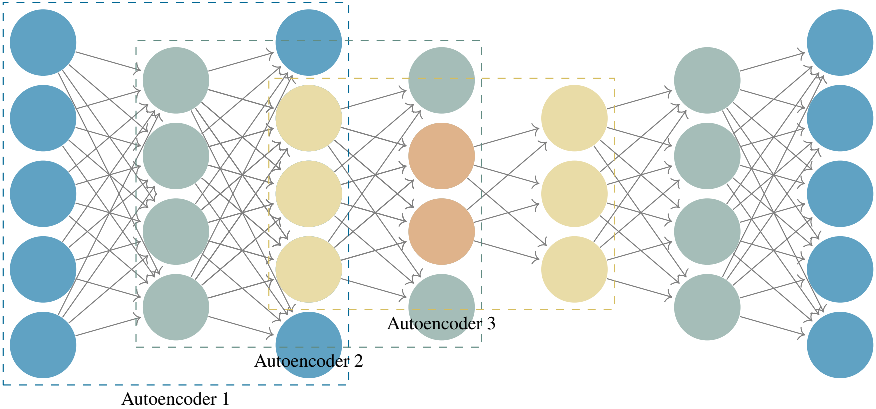
Sparse Autoencoders

Stacked Autoencoders (I)

- The standard AEs can learn to extract the fundamental features in the data.
- These are features of low level.
  - In the case of images, mainly edges and blobs.
- Is there a way to extract higher-level features?

- The solution is to stack several AEs to conform a **Stacked AE**.
  - A new AE is applied to the encoding obtain by a standard AE.
  - This procedure is repeated.
  - At each level an AE learns features from the features of the previous level.
  - The features become more and more abstract at each level.

Stacked Autoencoders (II)



Other Autoencoders

Supervised Stacked Autoencoders

1. The Stacked Autoencoders are trained in an **unsupervised way**.
2. A Deep NN is built using all the encoders together (first half of the network).
3. A dense layer is added at the end, and the whole NN is trained in a **supervised way** (fine tuning).

Deep Autoencoders

1. A Deep NN performs the encoding.
2. A Deep NN performs the decoding.
3. Both are trained as a whole in the usual "deep" way.
4. The architecture is often symmetric.

Notebook

Deep Autoencoders

Convolutional Neural Networks

Motivation

- ML models are usually sensitive with respect "translations" of the inputs.
  - A change like
$$\begin{pmatrix}0 & 1 & 3 & 5 & 0 & 0\end{pmatrix} \to \begin{pmatrix}0 & 0 & 1 & 3 & 5 & 0\end{pmatrix}$$
    will completely modify the prediction of the model.

- This type of perturbations are very common in certain problems, like **image recognition**.
  - Two images with the same face in a slightly different position should produce the same output.
  - This is not the case when using standard DNNs.

- **Convolutional Neural Networks** (CNNs) tackle this problem through a combination of:
  1. Convolution layers.
  2. Pooling layers.

Convolution

UAM

- A convolution is a linear transformation of a matrix **X** with a kernel (filter or mask) **K**.
- Mathematically, the convolution of $\mathbf{X} \in \mathbb{R}^{d \times d'}$ with the kernel $\mathbf{K} \in \mathbb{R}^{k \times k'}$ is a matrix $\mathbf{X} * \mathbf{K} \in \mathbb{R}^{(d-k+1)\times(d'-k'+1)}$ defined as:

$$(\mathbf{X} * \mathbf{K})_{i,j} = \sum_{a=1}^{k} \sum_{b=1}^{k'} (\mathbf{X})_{i+a-1,j+b-1} (\mathbf{K})_{a,b}, \quad 1 \le i \le d-k+1, \quad 1 \le j \le d'-k'+1.$$

$$\underset{\mathbf{X}}{\begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0_{\times 1} & 0_{\times 0} & 0_{\times 1} \\ 0 & 1 & 1 & 0 & 0_{\times 0} & 0_{\times 1} & 0_{\times 0} \\ 1 & 1 & 0 & 0 & 0_{\times 1} & 0_{\times 0} & 0_{\times 1} \end{pmatrix}} * \underset{\mathbf{K}}{\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}} = \underset{\mathbf{X} * \mathbf{K}}{\begin{pmatrix} 1 & 4 & 3 & 4 & 1 \\ 1 & 2 & 4 & 3 & 3 \\ 1 & 2 & 3 & 4 & 1 \\ 1 & 3 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 & \boxed{0} \end{pmatrix}}$$

Notebook

Convolution of Images

Convolution Layer

- The convolution is a powerful tool in image processing.
- Depending on the filter, a certain structure will be highlighted.

- Image convolution can be applied inside a NN through a **convolution layer**.
- The kernel matrix $\mathbf{K}$ corresponds to the **weights** of the layer.
  - They can be learnt during training.
  - The CNN will automatically find the filters that best suit the problem.
  - The number of weights is reduced with respect to a dense layer, preventing over-fitting.

- Multiple convolutions can be applied at once.
- A convolution layer will produce a tensor containing the input image processed with different filters.

Pooling

- The convolutional layer will produce images highlighting new features.
- These images can have a large dimension.
- A dense layer will still be sensitive to translations.
- A mechanism to make **space invariant** the features and **reduce the dimension** is needed.

- This can be done through **pooling**.
- An operation (usually the maximum or the average) is applied over disjoint patches of the images.

$$\begin{pmatrix} 1 & 1 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 3 & 2 & \boxed{\begin{matrix} 1 & 0 \\ 3 & 4 \end{matrix}} \end{pmatrix} \xrightarrow{\max} \begin{pmatrix} 6 & 8 \\ 4 & \boxed{4} \end{pmatrix} \qquad \begin{pmatrix} 1 & 1 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 3 & 2 & \boxed{\begin{matrix} 1 & 0 \\ 3 & 4 \end{matrix}} \end{pmatrix} \xrightarrow{\text{avg}} \begin{pmatrix} 3.3 & 5.3 \\ 2.0 & \boxed{2.0} \end{pmatrix}$$

Convolutional Neural Networks

- CNNs combine the convolution and pooling layers, usually in blocks with the structure:
  1. Convolution layer + ReLU.
  2. Pooling layer.
- This structure can be repeated several times with different dimensions.

- There are different architectures that have proven to be efficient in certain problems:
  - **LeNet**, a classic architecture for recognition of handwritten digits.

Layer	Input	1	2	3	4	5	6	Output
Type	Image	Conv.	Avg.Pool.	Conv.	Avg.Pool.	Conv.	Dense	Dense
Size	32×32	$6 \times 28 \times 28$	$6 \times 14 \times 14$	$16 \times 10 \times 10$	$16 \times 5 \times 5$	$120 \times 1 \times 1$	84	10

  - AlexNet, ResNet, GoogLeNet (more than 20 layers)...

- CNNs are used in many real applications.
  - Image and video recognition.
  - Recommendation systems.
  - Natural language processing.

Notebook

Convolutional Neural Networks

Recurrent Neural Networks

Motivation

- Most ML models assume that the samples $\{(\mathbf{x}_i, \mathbf{y}_i)\}$ are independent between them.
- This is not always the case, some problems are strongly context dependent.
  - Temporal series prediction.
  - Natural language processing.

- A naive approach is to modify directly the patterns to include such information using delays.
  - It requires to determine which delays to use.
  - It is a very limited approach.

- **Recurrent Neural Networks** (RNNs) tackle this problem through backwards connections.

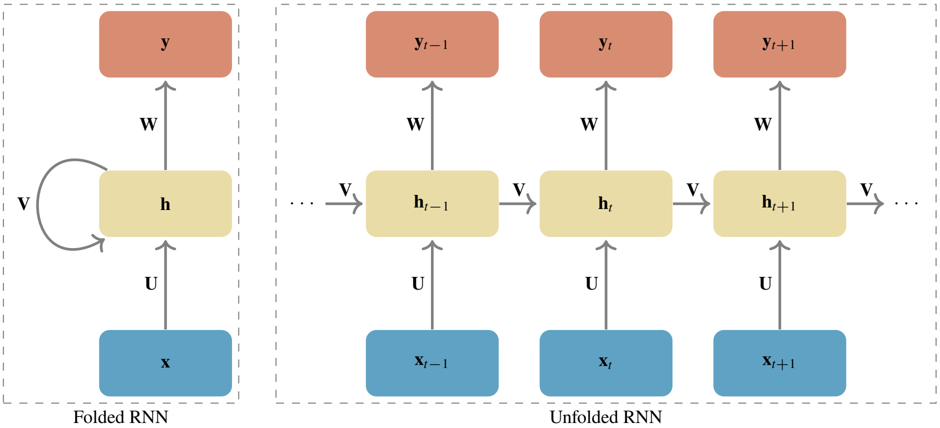
Recurrent Neural Networks (I)

- In a standard Feedforward NN there are no cycles in the connections.
  - The output only depends on the current input.
  - It is not aware of the state.

- An **RNN** has backwards connections.
  - Values previously produced by the network are plugged back as inputs.

- The RNNs cannot be trained using backpropagation, as there are cross dependencies between the weights.
- Instead, **Backpropagation Through Time** is used.
  1. The network is unfolded across time.
  2. Standard backpropagation is used to compute the gradient.
  3. The gradients with respect to each copy of the network are added together.

Recurrent Neural Networks (II)



Long Short Term Memory (I)

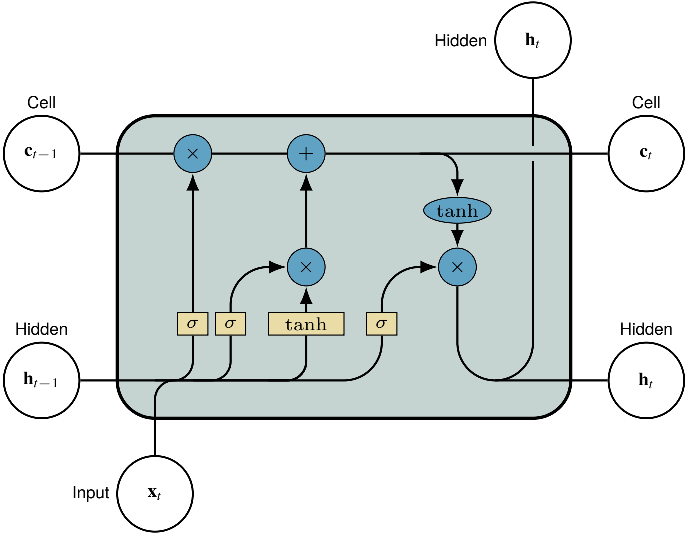
- The **Long Short-Term Memory** unit (LSTM) is a specialized component designed to retain information for larger intervals than standard RNN architectures.
- The LSTMs have several gates to control the memory of the network.
- These units are usually composed by the following elements:

  Cell Constitutes the memory of the LSTM unit.
  Input Gate Controls how much a new input flows into the cell.
  Forget Gate Controls how much a value remains in the cell.
  Output Gate Controls how much the value in the cell is used to compute the output the unit.

- There are connections into and out of the LSTM gates (some of them recurrent).

- The advantage of the LSTMs is that the weights of the gates are learnt during training.
- The network learns which pattern should be retained.

Long Short Term Memory (II)



Notebook

Recurrent Neural Networks

Generative Adversarial Networks

Motivation

- An unsupervised application of DNNs is to **generate new samples** following a learnt distribution.
  - For example, after being trained on many images of faces, the network should be able to generate a new (although realistic) face.
- Two questions arises:
  1. Which approach should be used to train such a network?
  2. Which inputs should the network use to generate new samples?

- The **Generative Adversarial Networks** answer both questions in an effective way.
  1. An adversarial approach, where two networks compete in a "game".
  2. Just random noise is enough.

Generative Adversarial Networks (I)

- A GAN is composed by two different networks: a **generative network** and a **discriminative network**.

Generative Network (Generator)

Input Points generated in a latent space, in practice multidimensional random noise.

Output Points generated from a data distribution of interest, in practice the generated samples.

Discriminative Network (Discriminator)

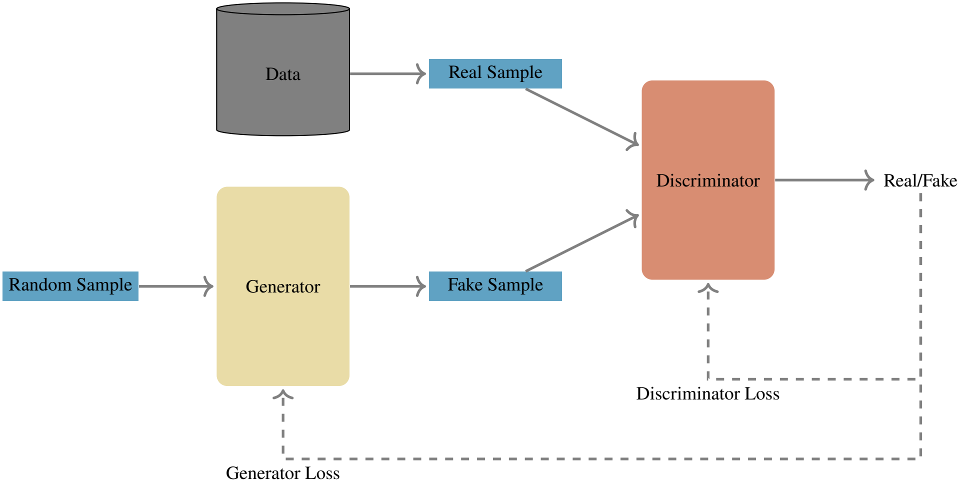
Classifies between samples generated by the generator and real samples from the data distribution of interest.

Training a GAN

The “magic” takes place in the adversarial training, where the following procedure is repeated:

1. The generator is used to generate synthetic samples.
2. The discriminator is trained to distinguish between the real and the generated samples.
3. The generator is trained to maximize the error of the discriminator.

Generative Adversarial Networks (II)



Notebook

Generative Adversarial Networks

Deep Neural Networks

Carlos María Alaíz Gudín