Autoencoders

Data Mining

Ester Vidaña Vila

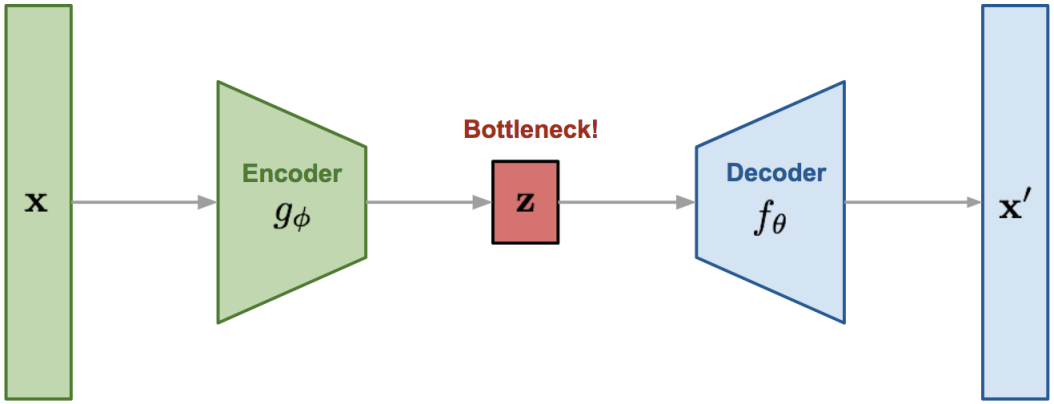
laSalle ENG
Universitat Ramon Llull

Autoencoders

Existen muchos tipos de redes neuronales, uno de ellos es el **Autoencoder**.

Los autoencoders son parte del aprendizaje no supervisado y disponen de una arquitectura que consta de tres partes.

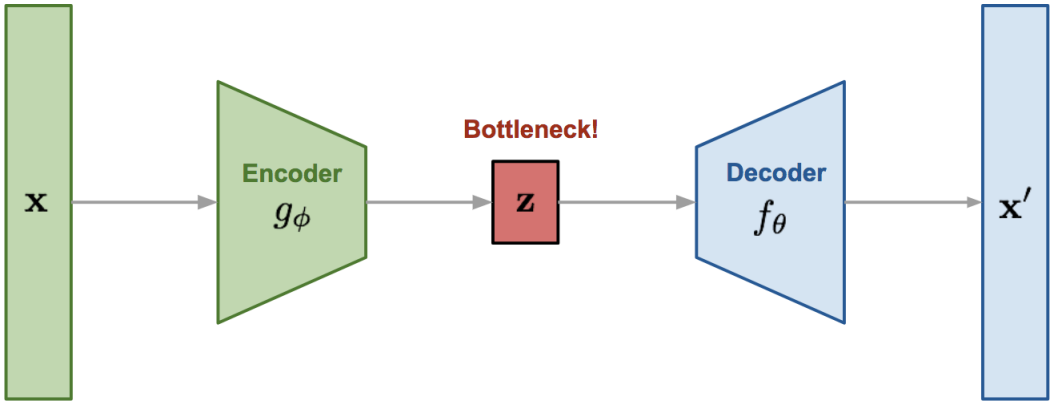
El Encoder, el Bottleneck y el Decoder.



Autoencoders

Este tipo de redes nos permiten tener una **representación compacta** de los datos que tengamos como entrada.

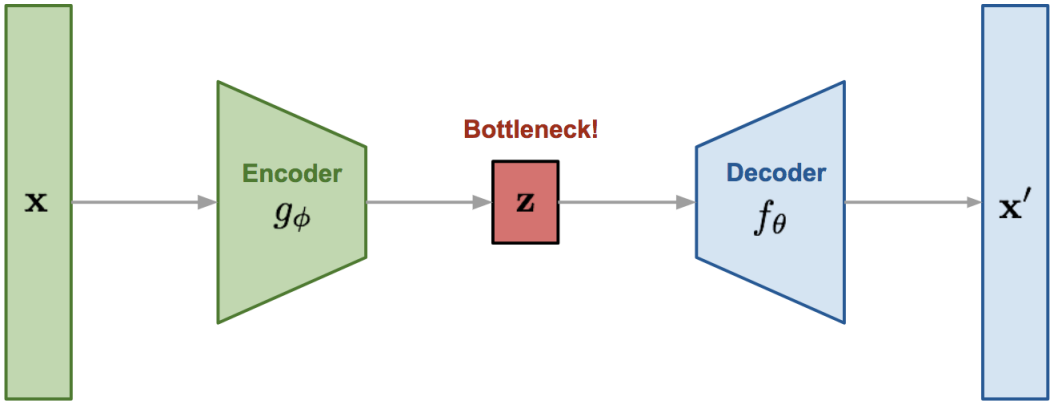
Esta representación la crea el módulo **Encoder**.



Autoencoders

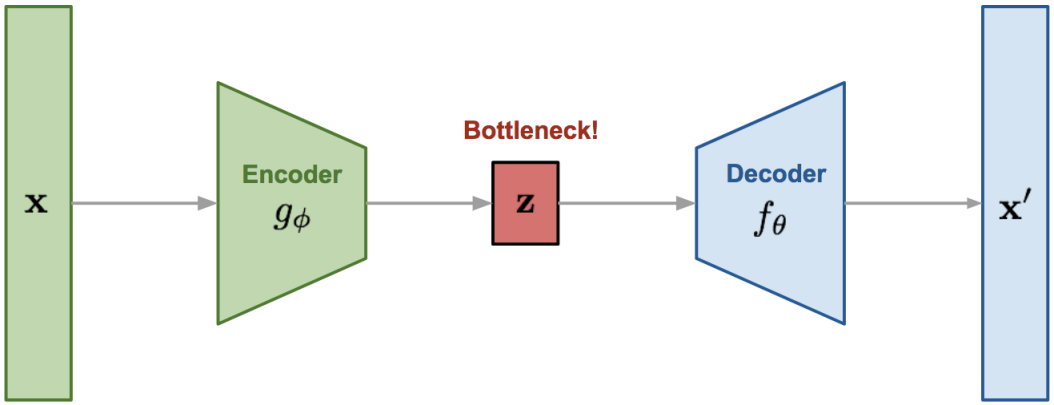
El **Bottleneck** es el resultado de la representación compacta del Encoder.

La representación que se obtiene en el Bottleneck se conoce como **Latent Space**.



Autoencoders

Por ultimo el **Decoder** es quien reconstruye la entrada a partir del Bottleneck.

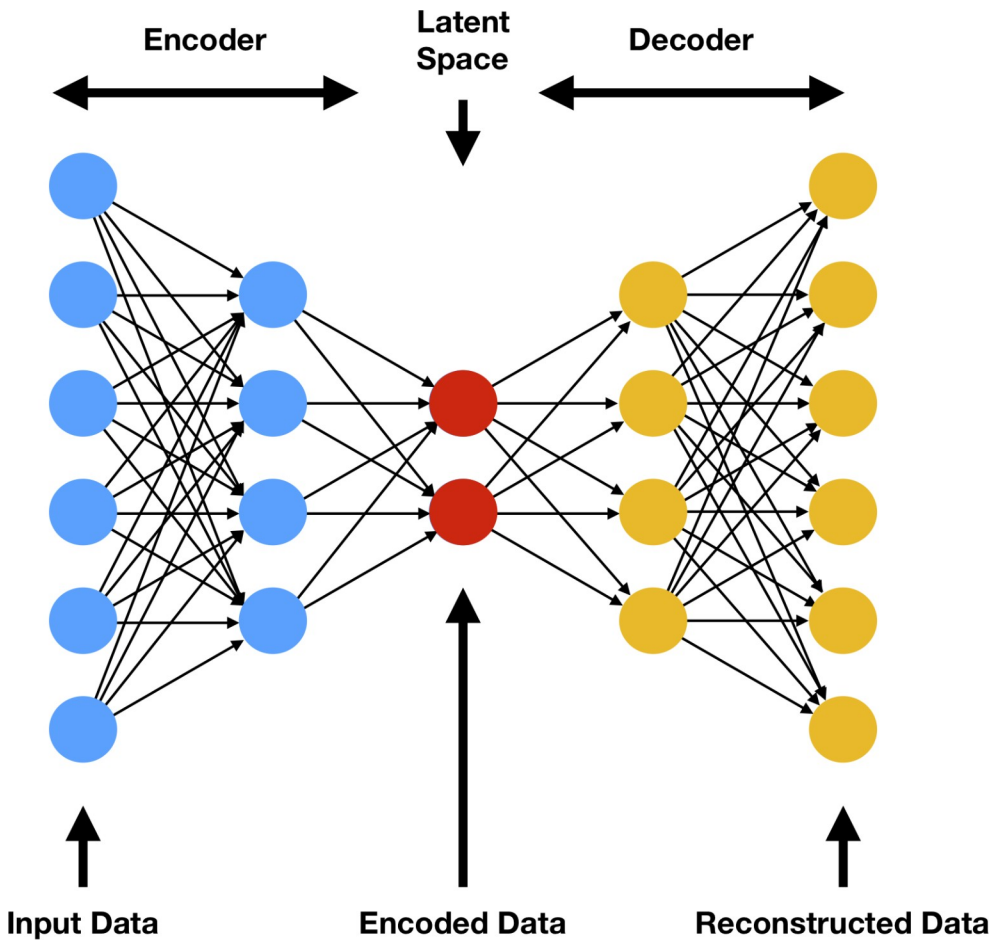


Autoencoders

Resumiendo:

- El Encoder comprimirá la información, aprendiendo a identificar la imformación más **relevante**.

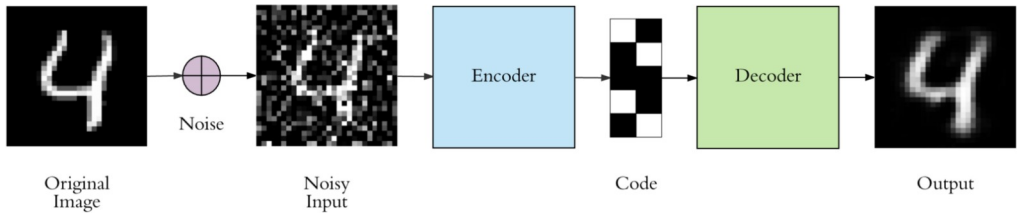
- El Decoder, en cambio, a partir de unas dimensiones reducidas, intentará obtener la misma salida que la entrada.



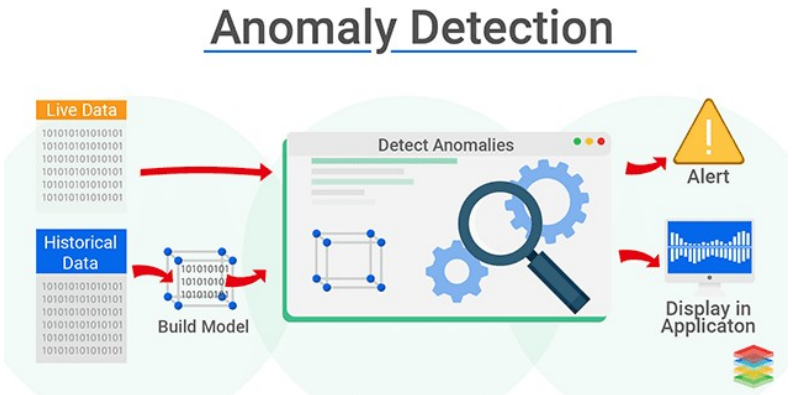
Autoencoders

Algunos casos de uso son:

Reconstrucción de imagenes



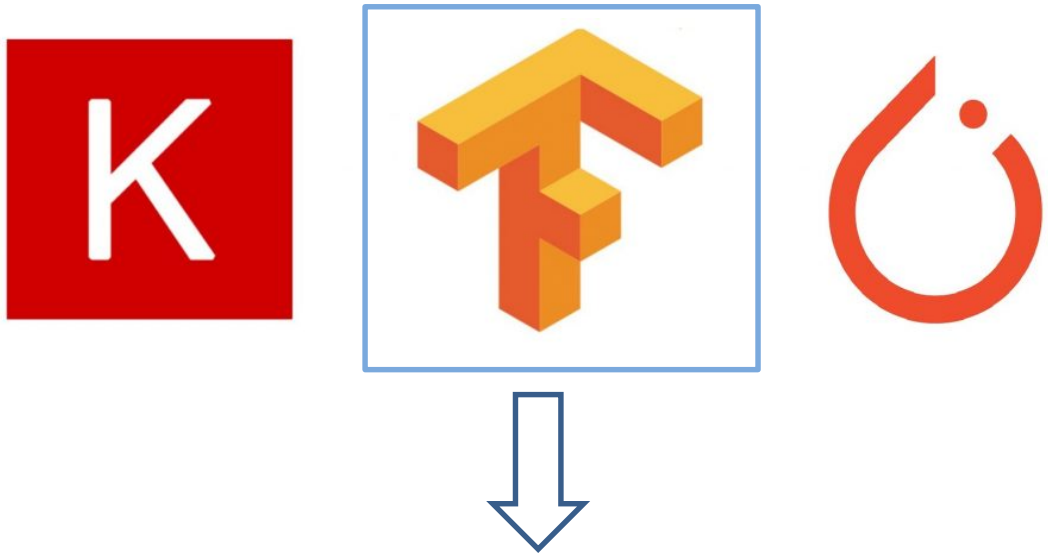
Detección de anomalías



Tensorflow vs. Keras vs. Pytorch

- Keras es una librería de redes neuronales de alto nivel programada en Python. Es *open-source* y permite correr experimentos sobre otros frameworks como Tensorflow, CNTK o Theano.
  - Se caracteriza por ser modular y user-friendly.
  - No se preocupa por los computos a bajo nivel, simplemente los subcontrata a las librerías de backend.

Tensorflow vs. Keras vs. Pytorch



- Tensorflow es un framework de Deep Learning que desarrolló Google en 2015.
  - Ofrece múltiples niveles de abstracción para construir y entrenar redes neuronales.
  - Tiene soporte sobre varias plataformas (e.g. Android).
  - Framework muy popular en la industria.

Tensorflow vs. Keras vs. Pytorch

- Pytorch es un framework de Deep Learning que desarrolló el equipo de investigación en AI de Facebook en 2017.
  - Fácil de utilizar, flexible.
  - Eficiente gestionando el uso de memoria.
  - Muy popular en la acadèmia (aunque también en la industria).

Tensorflow vs. Keras vs. Pytorch

	Keras	PyTorch	TensorFlow
API Level	High	Low	High and Low
Architecture	Simple, concise, readable	Complex, less readable	Not easy to use
Datasets	Smaller datasets	Large datasets, high performance	Large datasets, high performance
Debugging	Simple network, so debugging is not often needed	Good debugging capabilities	Difficult to conduct debugging
Does It Have Trained Models?	Yes	Yes	Yes
Popularity	Most popular	Third most popular	Second most popular
Speed	Slow, low performance	Fast, high-performance	Fast, high-performance
Written In	Python	Lua	C++, CUDA, Python

Fuente: https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article

Modelos en Keras corriendo sobre Tensorflow

- Hay dos formas de definir los modelos:
  - Modelos secuenciales:
    - Útil para modelos más sencillos, secuenciales.

```
model = tf.keras.Sequential()
model.add(tf.keras.Input(shape=(X_train.shape[1],)))
model.add(tf.keras.layers.Dense(20, activation='relu'))
model.add(tf.keras.layers.Dense(14, activation='relu'))
model.add(tf.keras.layers.Dense(20, activation='relu'))
model.add(tf.keras.layers.Dense(X_train.shape[1]))
```

  - Modelos funcionales:
    - Útil para crear modelos más complejos o que no sean secuenciales.

```
inputs = tf.keras.Input(shape=(X_train.shape[1],))
encoder = tf.keras.layers.Dense(20, activation='relu')(inputs)
latent_space = tf.keras.layers.Dense(14, activation='relu')(encoder)
decoder = tf.keras.layers.Dense(20, activation='relu')(latent_space)
outputs = tf.keras.layers.Dense(X_train.shape[1])(decoder)
model = tf.keras.models.Model(inputs=inputs, outputs=outputs)
```