

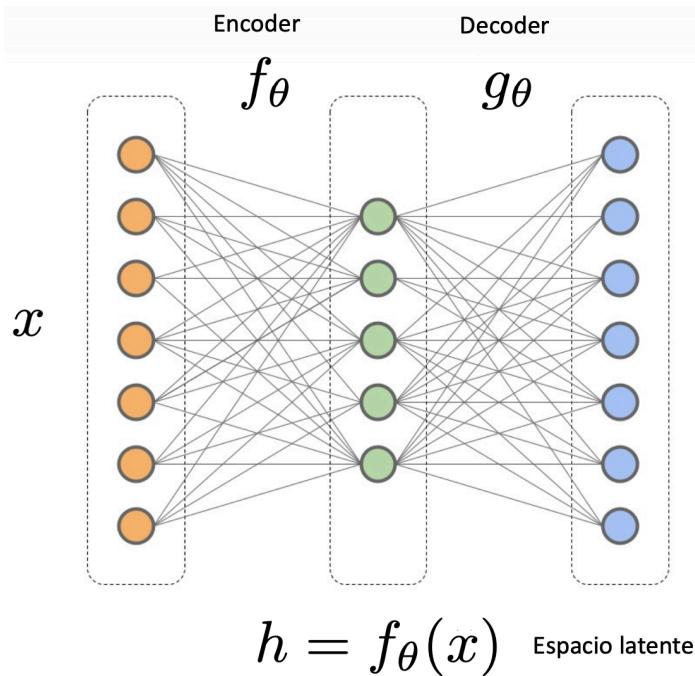
# Clase 6

## ¿Cómo aprender nuevas representaciones de los datos?

- Buscamos aprender nuevas representaciones (o embeddings) sin necesidad de utilizar etiquetas.

### Autoencoders

- Se separan en dos arquitecturas: una de esas estructuras codifica los datos (genera un embedding en menor dimensión) y la otra arquitectura los decodifica (i.e., lo devuelve a su estado y dimensión original).



- Lo que nos importa es quedarnos con el espacio latente, es decir, con la representación o embedding de menor dimensión.
- Este es un método **auto-supervizado**, porque no utilizamos etiquetas. La función de pérdida la computamos utilizando el propio dato.
  - Entonces, a la función de pérdida le pasamos la reconstrucción del decoder y lo comparamos con el valor de entrada. Al minimizar la función de pérdida nos aseguramos que la salida de nuestro modelo sea lo más parecida posible.

$$L(x; \theta) = (\underbrace{g_\theta(f_\theta(x))}_{\text{Decoder}} - \underbrace{x}_{\text{Imagen original}})^2$$

Encoder

- Lo que logramos es reducir dimensiones de la misma manera que PCA. La diferencia es que los autoencoders generan transformaciones no-lineales.

### Denoising Autoencoders

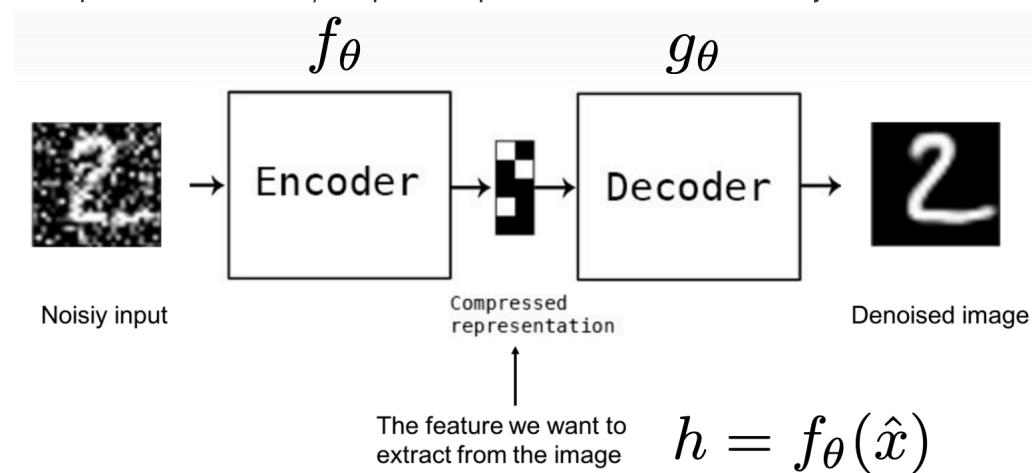
- La idea es agregar ruido a nuestros datos de entrenamiento. Al modelo lo entrenamos para que pueda devolver la imagen original, lo cual se logra modificando la función de pérdida.

$$L(x; \theta) = (\underbrace{g_\theta(f_\theta(\hat{x}))}_{\text{Decoder}} - \underbrace{x}_{\text{Imagen original}})^2$$

Imagen con ruido

Encoder

- La ventaja de este método es que podemos obtener un espacio latente (o embedding) que es más suave, lo que nos permite evitar el sobre-ajuste.



### ¿Para qué podemos usar autoencoders?

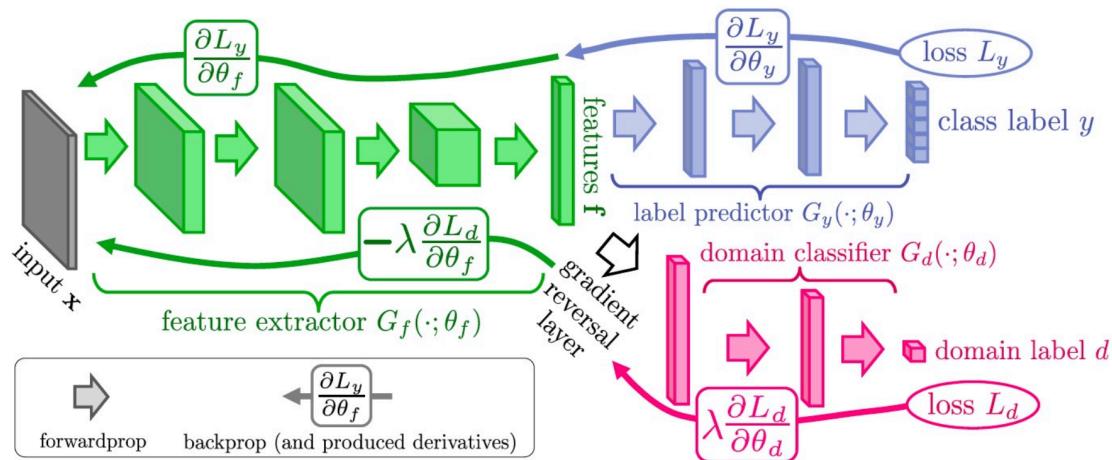
- Podemos utilizarlo para inicializar los pesos de nuestra red neuronal. Es decir, utilizamos el encoder para obtener representaciones (o embeddings) de menor dimensión para entrenar un MLP.
  - Lo bueno es que no necesitamos usar las etiquetas para aprender las representaciones de nuestros datos. Las etiquetas las utilizamos para entrenar el MLP.

- Si tenemos muchos datos, podemos seguir entrenando el encoder para ir mejorando los embeddings que aprendemos. Si no tenemos tantos datos, solo entrenamos la MLP.
- Para detección de anomalías.
  - Entrenamos nuestro autoencoder con los casos normales (e.g., los casos que no son fraude).
  - Computamos el error de reconstrucción para estos casos normales (con los que entrenamos el modelo). También computamos el error de reconstrucción para los datos que son fraude (con los que no entrenamos).
  - El error de reconstrucción lo utilizamos para elegir el corte. Si para una observación el error de reconstrucción es mayor al error de reconstrucción para los casos normales, lo clasificamos como una anomalía.
- Para post-procesamiento.
  - Entrenamos un autoencoder con una tarea de pretexto. Por ejemplo, entrenamos un autoencoder para aprender representaciones de imágenes de pulmones (paper de Enzo).
  - Luego, segmentamos las imágenes con un modelo cualquiera (e.g., un Random Forest). Luego, utilizamos el autoencoder para que reconstruir la salida de nuestro modelo inicial.

## ¿Cómo generalizar múltiples dominios?

- Esto ocurre cuando entrenamos nuestro modelo con, por ejemplo, imágenes de números escritos a mano y ahora tenemos que clasificar imágenes de números de una casa.
  - Si al momento de entrenar el modelo conocemos los distintos dominios, y tenemos etiquetas para todos ellos, podemos entrenar el modelo con todos esos datos.
  - Si no tenemos información sobre los múltiples dominios de antemano, tenemos dos opciones:
    - Si tenemos las etiquetas para el nuevo dominio, realizamos un fine-tuning del modelo.
    - Si no tenemos las etiquetas para el nuevo dominio, lo que podemos hacer es mapear la distribución de los datos de un dominio a otro.
- El problema de entrenar un modelo con datos de un único dominio es que las features que utilizamos pueden no ser explicativas para otro dominio. Una posible solución es utilizar **features invariantes**, las cuales son agnósticas al dominio. Para eso, vamos a entrenar una red convolucional que extraiga las features, las cuales usamos para entrenar dos redes:

- La primer red la entrenamos para clasificar imágenes utilizando las etiquetas. En ese caso solo vamos a tener etiquetas para un único dominio.
- La segunda red (que llamamos **red adversaria**) la entrenamos para que pueda distinguir si una imagen viene del primer o segundo dominio.
  - La función de pérdida la definimos para que "confunda" a la red. Esto quiere decir que si la red se confunde de dominio, mejor. Eso nos va a estar diciendo que las features que estamos usando son agnósticas al dominio.
  - Cuando hacemos back-propagation, ajustamos los pesos de la red convolucional para que gener features más agnósticas, es decir, que no sean dependientes del dominio.



## ¿Como interpretar los modelos?

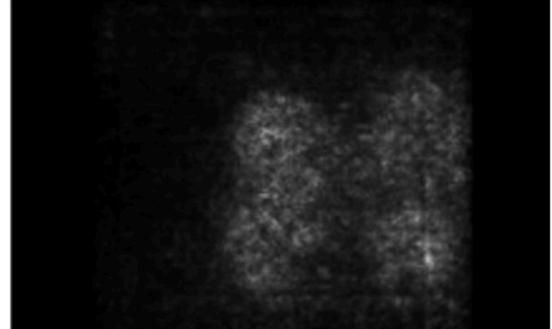
### Mapas de saliencia por medio de oclusión

- Miramos las áreas de la imagen que son más importantes para realizar la predicción. La idea es tapar un área de la imagen y mirar como cambia la predicción de nuestra red neuronal.
  - Si la predicción cambia mucho, entonces ese área es muy importante para la clasificación.
  - Si la predicción cambia poco, entonces ese área no es muy importante
- Con eso podemos armar un mapa de oclusión (mapa de calor) que indica que área es la más importante para el modelo.

### Mapas de saliencia por retropropagación

- Al realizar back-propagation no calculamos el gradiente de la función de pérdida con respecto a los pesos, sino con respecto a los valores de entrada. Esto nos permite ver cuánto cambia la función de pérdida cuando cambiamos nuestros valores de entrada.

- Tomamos el valor absoluto del gradiente. No nos importa en que dirección cambia la función de pérdida, nos importa que cambie.



## ¿Como evitar el sesgo?

- Sesgo algorítmico: cuando el modelo falla sistemáticamente para un grupo demográfico específico.
- **Atributos protegidos.** Features que, en principio, no queremos que el modelo utilice para tomar decisiones.