UNIVERSIDAD NACIONAL
DEL NORDESTE

CONICET

I M I T

Modulo 7:

Deep learning - Autoencoders - Variational autoencoders

Objetivos

- Arquitecturas profundas en imágenes.
- Autocodificadores/autoencoders
- UNET
- Modelos generativos:
  - GAN
  - Variational autoencoders
  - Flujos normalizantes

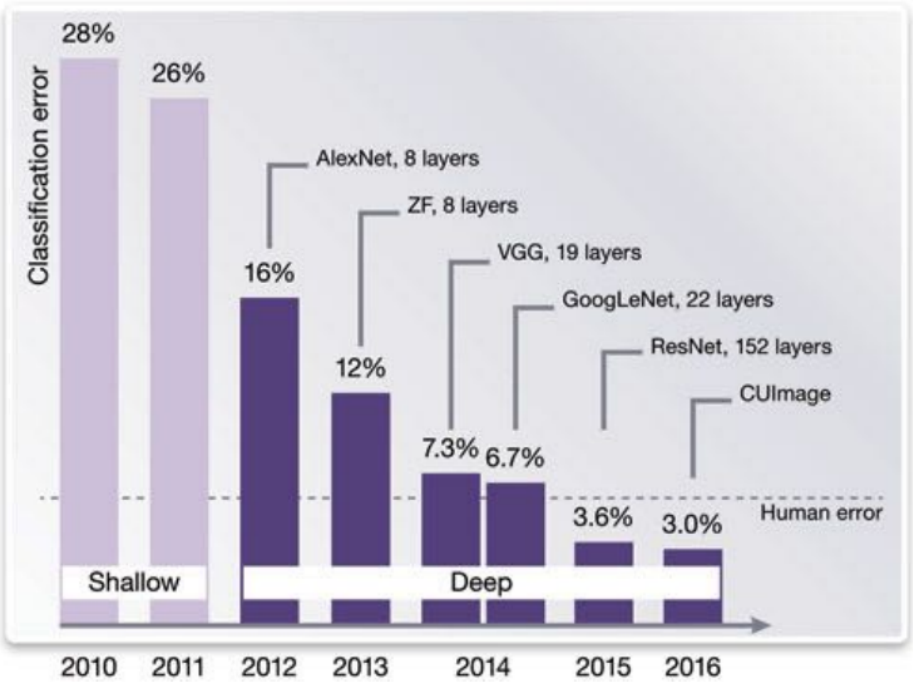
Bibliografía

- Goodfellow, I., Bengio, Y. and Courville, A., 2016. Deep learning. MIT press.
- Zhang, A., Lipton, Z.C., Li, M. and Smola, A.J., 2021. Dive into deep learning. arXiv preprint arXiv:2106.11342.

Historia de la clasificacion de imágenes

- LeNet-5 (1998) LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., & others. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278-2324.
- AlexNet (2012) Kizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in NIPS, 1097-1105.
- ZFNet (2013) Zeiler, M.D. and Fergus, R., 2014. Visualizing and understanding convolutional networks. In Computer Vision-ECCV 2014
- VGGNet (2014) Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556.
- GoogLeNet (2014) Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V. and Rabinovich, A., 2015. Going deeper with convolutions. In Proceedings of the IEEE
- ResNet (2015) He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In Proceedings of the IEEE 770-778. 170.000 citas!

Papers mas relevantes de Image Recognition



Fuente Reddit.

- Todos los papers basados en la base ImageNet.
- Importantes avances en la performance en reconocimiento de objetos.
- La performance se correlaciona con el aumento de capas (mas profundo mas performance).
- ResNet tiene menor error que un humano!

Herramientas de la convolución 1: Padding

Dado un vector de datos cuando comenzamos y cuando terminamos no tenemos datos para generar la convolución de la primera componente del vector:

$$y_j = \sum_{i=-N_k//2}^{N_k//2} w_{i+N_k//2} \quad x_{j+i}$$

Que pasa para $j = 0$, $N_k = 5$?, necesito x_{-2}??

- Agrego $N_k//2$ puntos con 0s: $x_{-2} = x_{-1} = 0$, $x_{N_x} = x_{N_x+1} = 0$.
- Asumo dominio cíclico o periódico $x_{-1} = x_{N_x-1}$, $x_{-2} = x_{N_x-2}$, $x_{N_x} = x_0$
- Asumo dominio espejo: $x_{-1} = x_1$, $x_{-2} = x_2$
- Reduzco componentes del vector de salida en $2 \times N_k//2$:

$$y_0 = \sum_{i=-N_k//2}^{N_k//2} w_{i+N_k//2} x_{j+i+N_k//2}$$

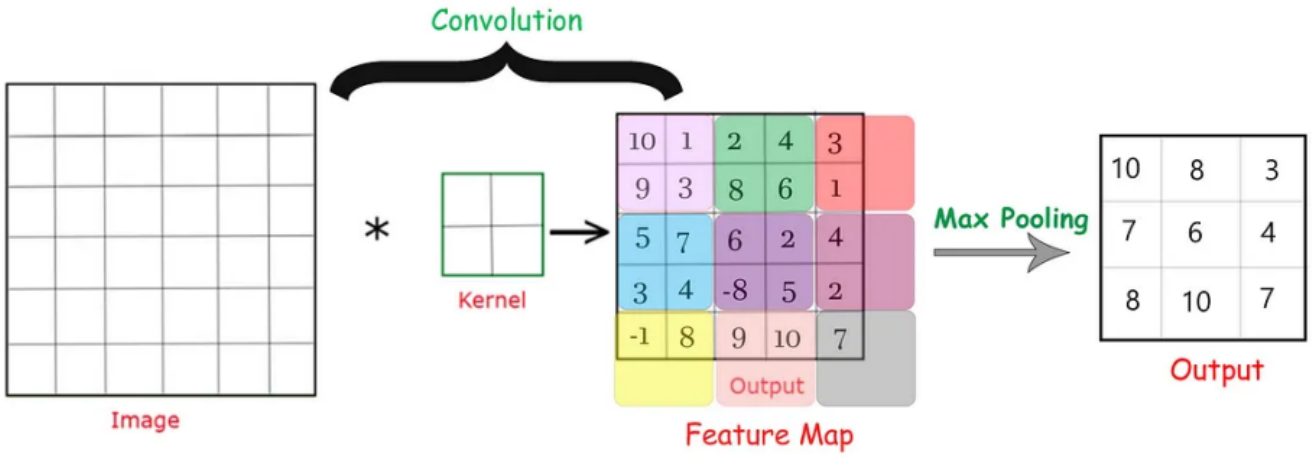
2. Downsampling

¿Como disminuyo la resolución de las imágenes?
Supongamos que quiero ir de 256 pixels a 128. Manteniendo el mismo dominio/apertura.

- Puedo ir saltando de dos pixels y me quedo entonces con los pares (stride). En dos dimensiones son rectángulos.
- Tomo el máximo entre dos pixels o en el rectángulo de dos por dos (max-pooling).
- Promedio los pixels del rectángulo: Average pooling

Lo debo realizar para cada canal por separado.

2. Max-pooling

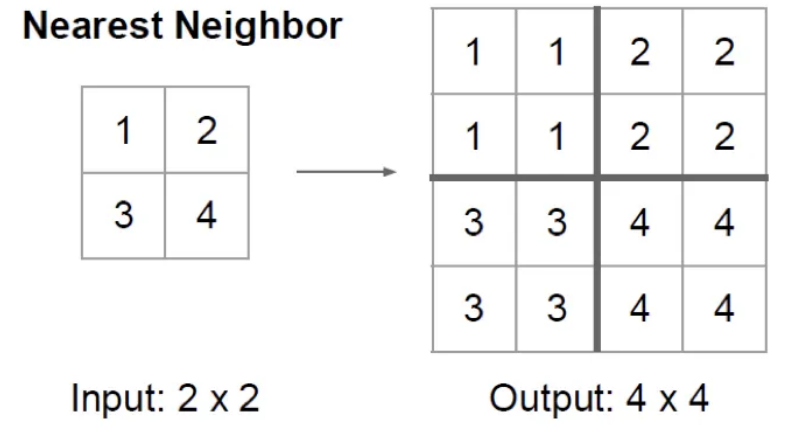


- Invariancia en la posición.
- Invariancia en la rotación.
- Invariancia a la escala (chico o grande)
- Selecciona las características mas importantes de la entrada.

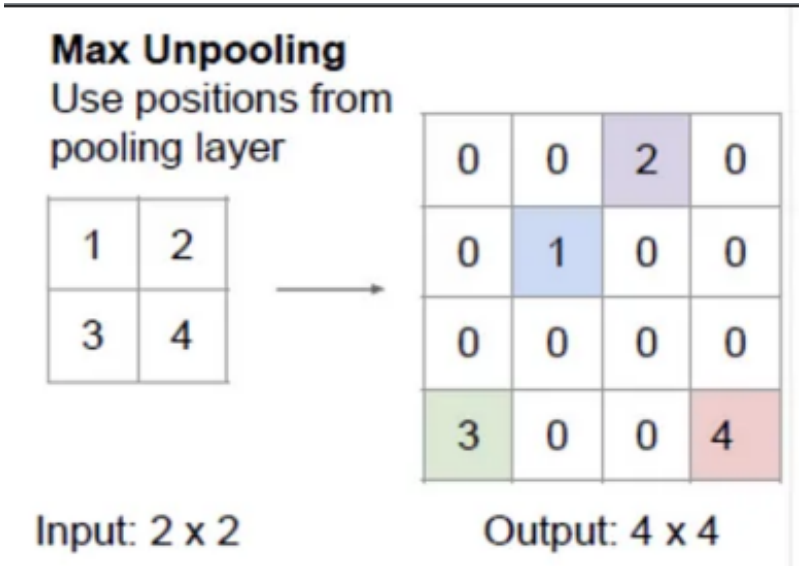
- Pierde información. El average pooling conserva mas información.
- Puede causar sobre-suavizado de las características, con pérdida de detalles de peq. escala.

2. Upsampling

Aumento de la resolución de las imágenes. Tengo que inventar un pixel en el medio de dos que ya tengo.

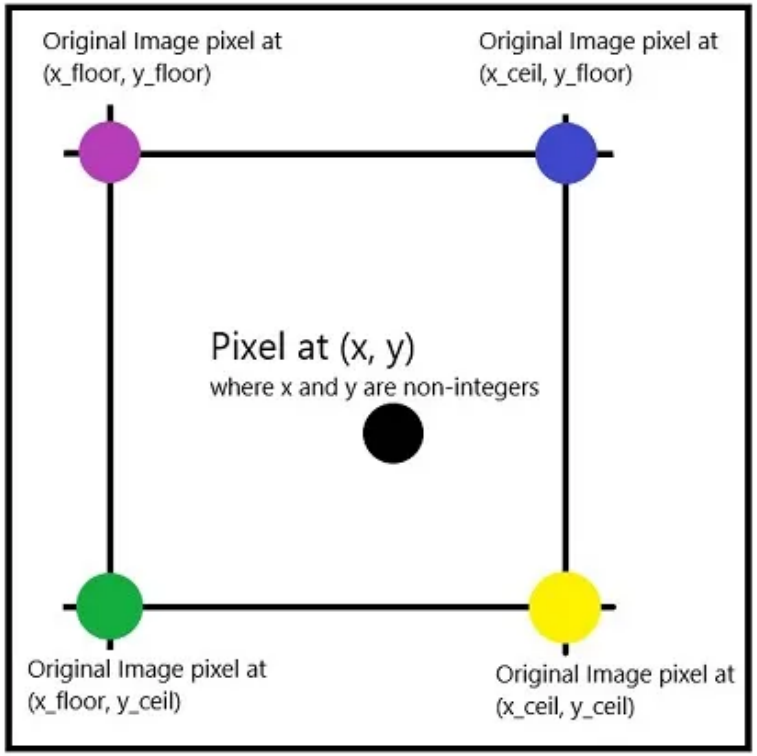


Copio o replico cada pixel. En 2d aparte del original son 3!.

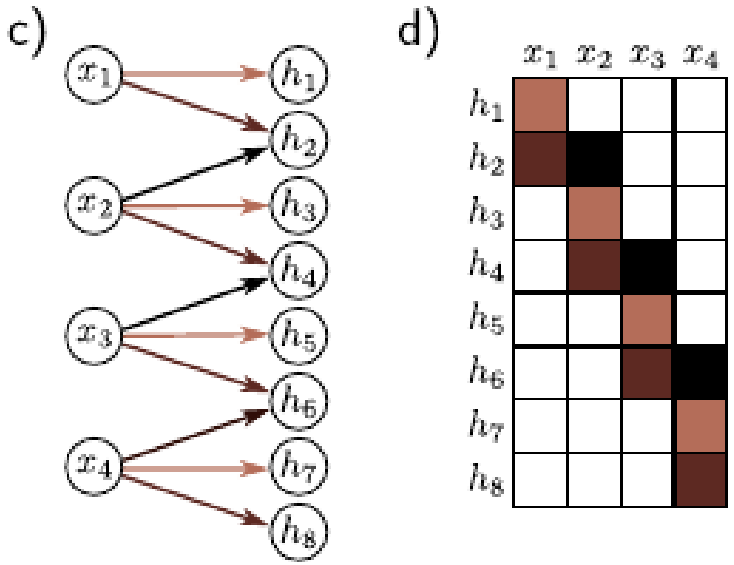


Max-unpooling. Tengo que registrar donde estaba el máximo.

2. Upsampling

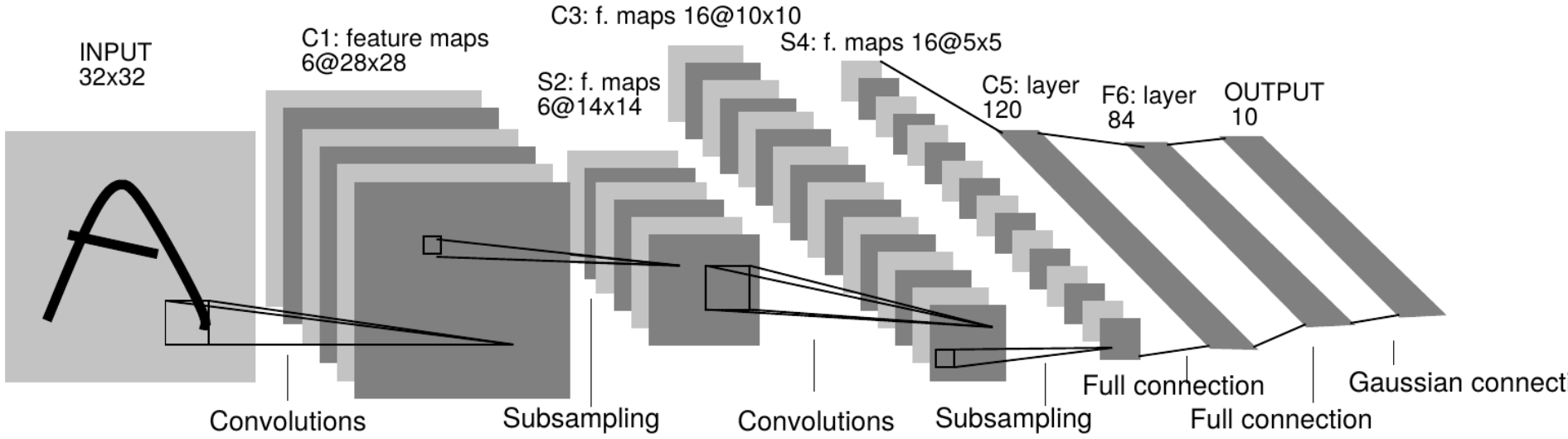


Interpolación bilineal (de los vecinos cercanos).



Transpose convolución.

Precursor: LeNet-5



Desarrollada por varios años para digitalizar los códigos postales a partir de las cartas. También se utiliza en cajeros.

- Kernels 5x5
- El subsampling se hacia con la suma de los 4 vecinos. $\sigma[(\alpha \sum_{n=1}^{4} C_n + \beta)]$. Luego se aplica una sigmoide.
- No se aplican todas las conexiones entre S2 y C3.
- Entre S4 y C5 hay una FC. Tangente hiperbólica como función de activación.
- La última capa se compone de unidades de RBFs con 84 inputs c/u. La salida de la RBF es $y_i = \sum_i (x_j - w_{ij})^2$.

La bisagra: Alexnet

Aplicación: ImageNet Large Scale Visual Recognition Challenge

ImageNet: base de datos con mas de 14 millones de images de alta resolución clasificadas con 22.000 clases. (web scrapping).

https://image-net.org/

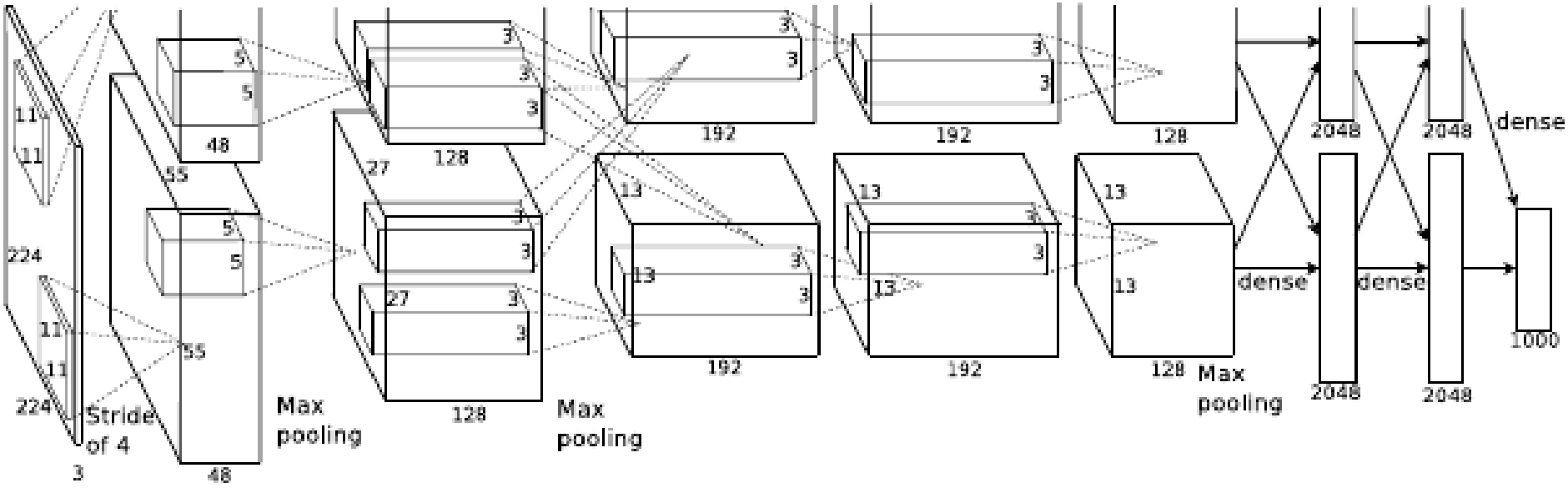
La imágenes que se usaron son de 224x224x3.

Implementada en GPUs (2 x GTX 580).

La profundidad del modelo es esencial para obtener alta performance.

Kizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in NIPS, 1097-1105.

Arquitectura Alexnet



8 capas. La división en dos es por las GPUs.

5 primeras con convolucionales. Algunas con maxpooling.

Las últimas tres son FC.

$$(CNN \rightarrow RN \rightarrow MP)^2 \rightarrow (CNN^3 \rightarrow MP) \rightarrow (FC \rightarrow DO)^2 \rightarrow Linear \rightarrow softm$$

CNN = capa convolucional (con funcion de activacion ReLU)

RN = normalización local

MP = max-pooling

FC = capa fully connected (con funcion de activacion ReLU)

Linear = capa fully connected (sin activacion)

DO = dropout

¿Porque Alexnet fue un landmark?

Resultados: AlexNet ganó la competición de ImageNet 2012 con un error top-5 de 15.3%, comparado al 2do lugar que obtuvo 26.2% (por afano).

- Activación con ReLU. AlexNet usaba ReLU en lugar de la tanh. Las ReLUs se pueden entrenar mucho mas eficientemente.
- Overlapping Pooling. Evaluan el overlap en la aplicación de las CNNs disminuye el error y encuentran menos overfit.
- Batch size of 128. Algoritmo de aprendizaje: SGD Momentum
- Problema de overfitting: AlexNet tenia 60 millones de parámetros (gracias a las 2 GPUs de 3gb), para disminuir el overfitting se uso:
  - Data Augmentation. Se generaron imágenes a través de reflecciones horizontales y traslaciones.
  - Dropout. Uno de los primeros modelos de producción en aplicar el dropout.

Alexnet in pytorch

```
class AlexNet(nn.Module):
    def __init__(self, num_classes: int = 1000, dropout: float = 0.5) -> None:
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=11, stride=4, padding=2),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=3, stride=2),
            [[VARIAS MAS]]... )
        self.avgpool = nn.AdaptiveAvgPool2d((6, 6))
        self.classifier = nn.Sequential(
            nn.Dropout(p=dropout),
            nn.Linear(in_features=256 * 6 * 6, out_features=4096),
            nn.ReLU(inplace=True),
            nn.Dropout(p=dropout),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, num_classes)  )
    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

Tamaño del kernel

En lugar de tener kernels en las convoluciones de 11x11, 5x5, 3x3.
Se puede tener múltiples kernels de 3x3 y el efecto es el mismo.
Dos de 3x3 tendrían la misma cobertura y “localidad” que uno de 5x5.
En el caso del max-pooling se puede usar un max-pooling de 2x2.

Es una de las fuentes de inspiración de la VGGNet 138 millón de paramétros

Normalización del batch

Las entradas $\mathbf{x}$ se asumen que tienen norma finita. Y generalmente uno procesa las variables de entrada para que tengan media 0 y la identidad como covarianza.

En redes profundas, la entrada de la activación se transforma a través de la media $\boldsymbol{\mu}_{\mathcal{B}}$ y la varianza $\boldsymbol{\sigma}_{\mathcal{B}}$ del batch $\mathcal{B}$ a través de

$$BN(\mathbf{x}) = \boldsymbol{\gamma} \odot \frac{\mathbf{x} - \boldsymbol{\mu}_{\mathcal{B}}}{\boldsymbol{\sigma}_{\mathcal{B}} + \epsilon} + \boldsymbol{\beta}$$

γ y β son hiperparámetros.

$$\boldsymbol{\mu}_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} \sum_{x \in \mathcal{B}} \mathbf{x} \quad y \quad \boldsymbol{\sigma}^2_{\mathcal{B}} = \frac{1}{|\mathcal{B}|} (\mathbf{x} - \boldsymbol{\mu}_{\mathcal{B}})^2$$

BN es útil como una forma de regularización y mejora la convergencia durante la optimización. $\mathcal{B}$ debería contener como mínimo 20 elementos, optimo 50-100. Ioffe, S., & Szegedy, C. (2015). Batch normalization: accelerating deep network training by reducing internal covariate shift. arXiv preprint

Normalización de los pesos

Normalización del batch tiene algunos problemas:

- Solo se puede usar con tamaños de mini-batch grandes >32.
- No funciona con redes recurrentes, el batch cambia.
- No queda claro como usarlo en prediccion/inferencia. Generalmente los valores medios obtenidos en el entrenamiento.

Salimans, T. and Kingma, D.P., 2016. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. Advances in neural information processing systems, 29.

Se usan dos parametros g (longitud del vector de pesos) y v para "normalizar" los pesos en el descenso de gradientes:

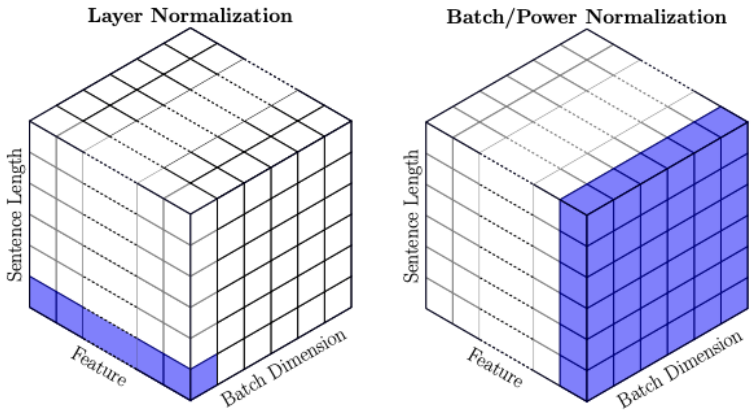
$$\mathbf{w} = \frac{g}{|\mathbf{v}|}\mathbf{v}$$

Acelera el, entrenamiento como la BN sin embargo en la práctica es

Normalización de la capa

Normalización del batch tiene algunos problemas:

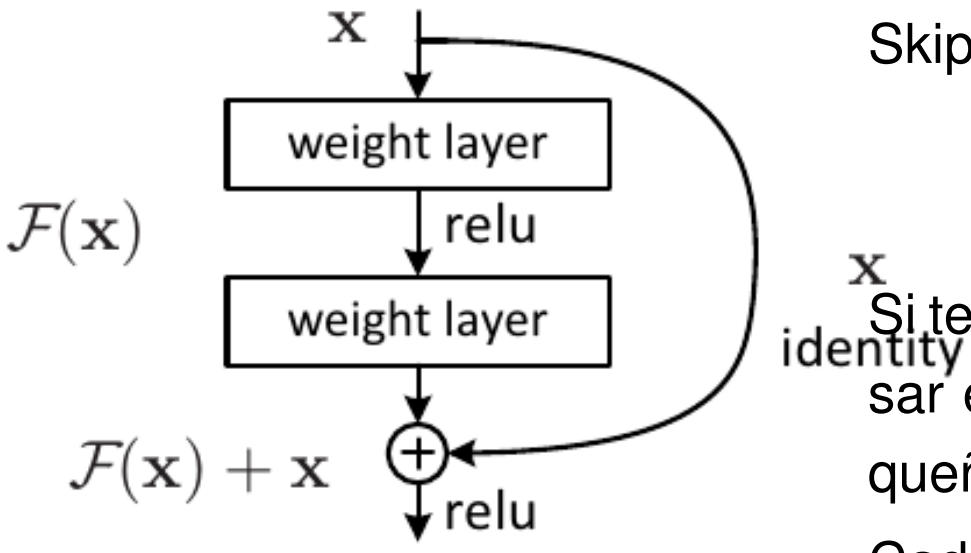
- Solo se puede usar con tamaños de mini-batch grandes >32.
- No funciona con redes recurrentes, el batch cambia.
- No queda claro como usarlo en prediccion/inferencia. Generalmente los valores medios obtenidos en el entrenamiento.



Se normaliza las características a media zero y varianza unitaria en cada capa antes de aplicar la función de activación.

Ba, J. L., Kiros, J. R., & Hinton, G. E. (2016). Layer normalization. arXiv preprint arXiv:1607.06450.

Resnet



Sumo a la salida de la red/bloque la entrada.

Skip connections - identity mappings:

$$\mathbf{y} = f(\mathbf{x}) + \mathbf{x}$$

Si tenemos múltiples capas, puedo pensar en cada una es una corrección pequeña a la anterior.

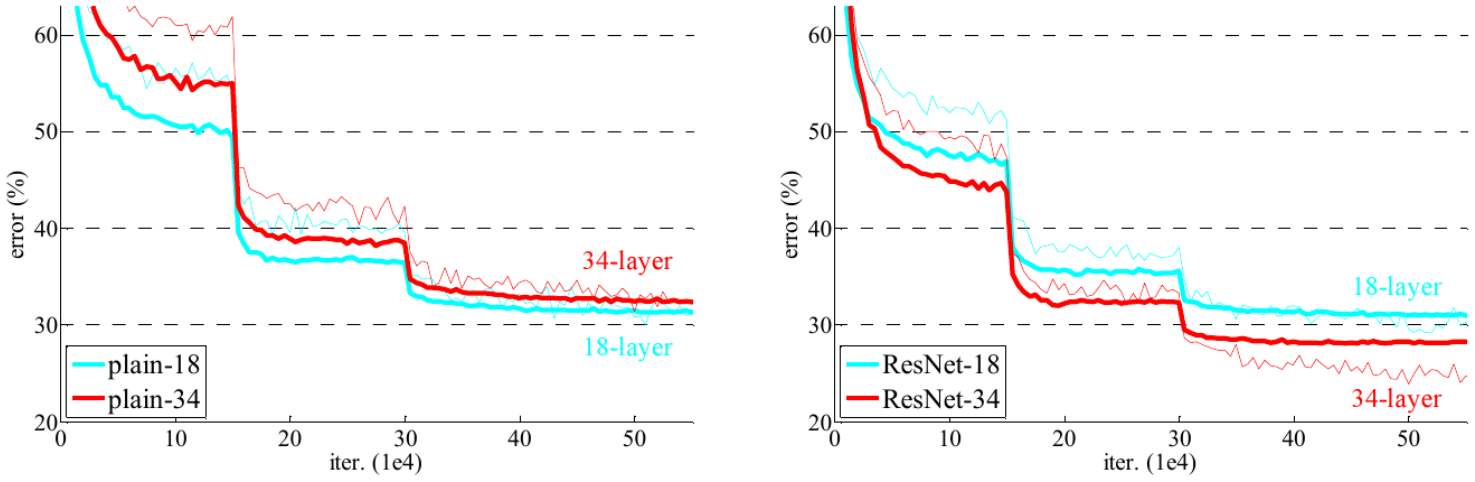
Cada capa adicional contiene la función de identidad.

De esta manera el nuevo modelo es tan bueno como el anterior con una corrección para reducir el error. Salto enorme en la interpretabilidad

La forma matemática coincide con los mapas: $\mathbf{x}_k = \mathbf{x}_{k-1} + f(\mathbf{x}_{k-1})$

Esquema de Euler $\frac{\mathrm{d}x}{\mathrm{d}t} = \tilde{f}(x) \rightarrow \mathbf{x}_k = \mathbf{x}_{k-1} + \Delta t \tilde{f}(\mathbf{x}_{k-1})$

Red residual o salto de conecciones



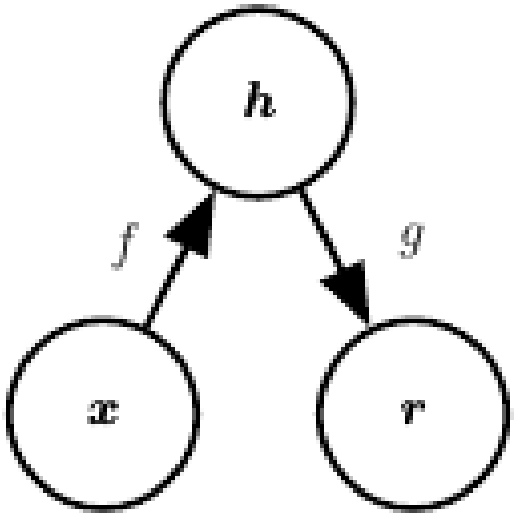
La ResNet profunda (34 capas) mejora significativamente a la de 18.

Esto no sucede con la estructura de la AlexNet.

Bloque residual en pytorch

```
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(ResidualBlock, self).__init__()
        self.conv1 = conv3x3(in_channels, out_channels, stride)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(out_channels, out_channels)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.downsample = downsample

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        if self.downsample:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out

model = ResNet(ResidualBlock, [2, 2, 2]).to(device)
```

Autocodificadores



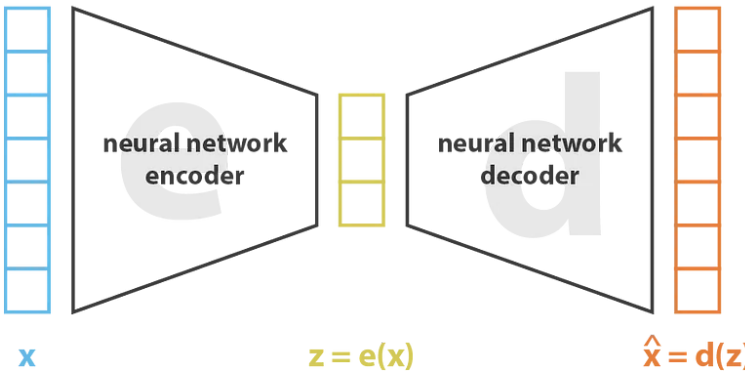
Idea que en principio parece absurda:
Quiero transformar la input $\mathbf{x}$ con f a un espacio latente, $\mathbf{h} = f(\mathbf{x})$ y luego anti-transformar con g de vuelta al espacio de la input $\mathbf{r} = g(\mathbf{x})$.

$\rightarrow$ La clave esta en la dimensionalidad: $\mathbf{x} \in \mathbb{R}^{N_x}$, $\mathbf{h} \in \mathbb{R}^{N_h}$ y $\mathbf{r} \in \mathbb{R}^{N_x}$, asumimos que $N_h \ll N_x$

Técnica NO supervisada. No requerimos de “targets”.

Obligo a ir a un espacio latente pequeño y a representar todo lo posible en ese espacio. Si no reduzco estoy copiando.

Autocodificadores



$$\text{loss} = \| x - \hat{x} \|^2 = \| x - d(z) \|^2 = \| x - d(e(x)) \|^2$$

- La complejidad de f también es central para obligar a la codificación.
- Si f y g son lineales resulta en PCA.
- El autoencoder es una técnica de reducción de la dimensionalidad nolineal = PCAs nolineales.

Queremos aprender a replicar las inputs, pero aprendiendo de la esencia de $\mathbf{x}$ en $\mathbf{h}/\mathbf{z}$.

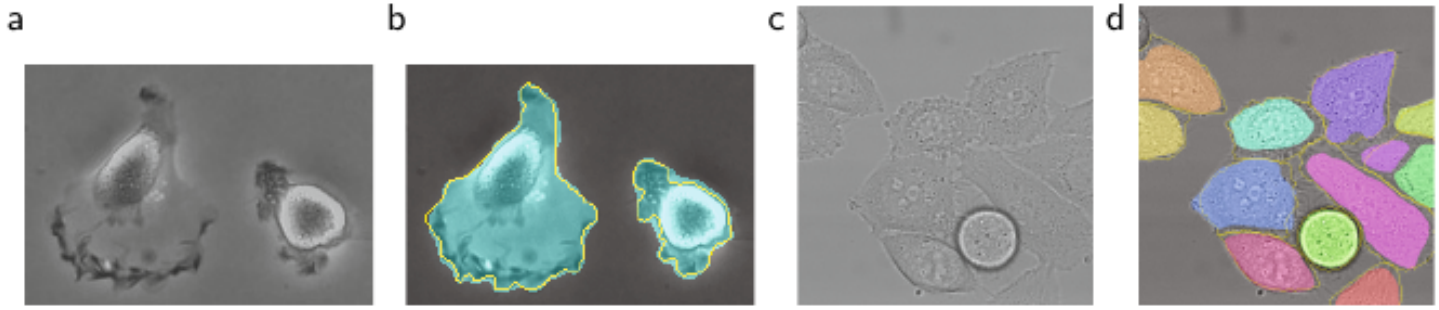
Autocodificadores

Para evitar la memorización de las inputs, $N_h \ll N_x$ con modelos complejos podemos regularizar.

- Regularización:
  - representación esparsa $J(\mathbf{x}, g(f(\tilde{\mathbf{x}}))) + \lambda \sum_i \|\nabla_{\mathbf{x}} h_i\|^2$,
  - derivada pequeña $J(\mathbf{x}, g(f(\tilde{\mathbf{x}}))) + \lambda \sum_i \|\nabla_{\mathbf{x}} h_i\|^2$,
  - robustocidad al ruido.
- Modelos generativos: autocodificadores variacionales, redes generativas estocasticas. Aprenden a maximizar la probabilidad (ELBO) evitando la memorización.
- Autocodificadores para denoising. $\tilde{\mathbf{x}} = \mathbf{x} + \boldsymbol{\epsilon}$, función de pérdida: $J(\mathbf{x}, g(f(\tilde{\mathbf{x}})))$.

UNETs

Reconocimiento visual. Clasificación de imágenes.



Aplicaciones: imagenes biomédicas (estructura multiresolución).

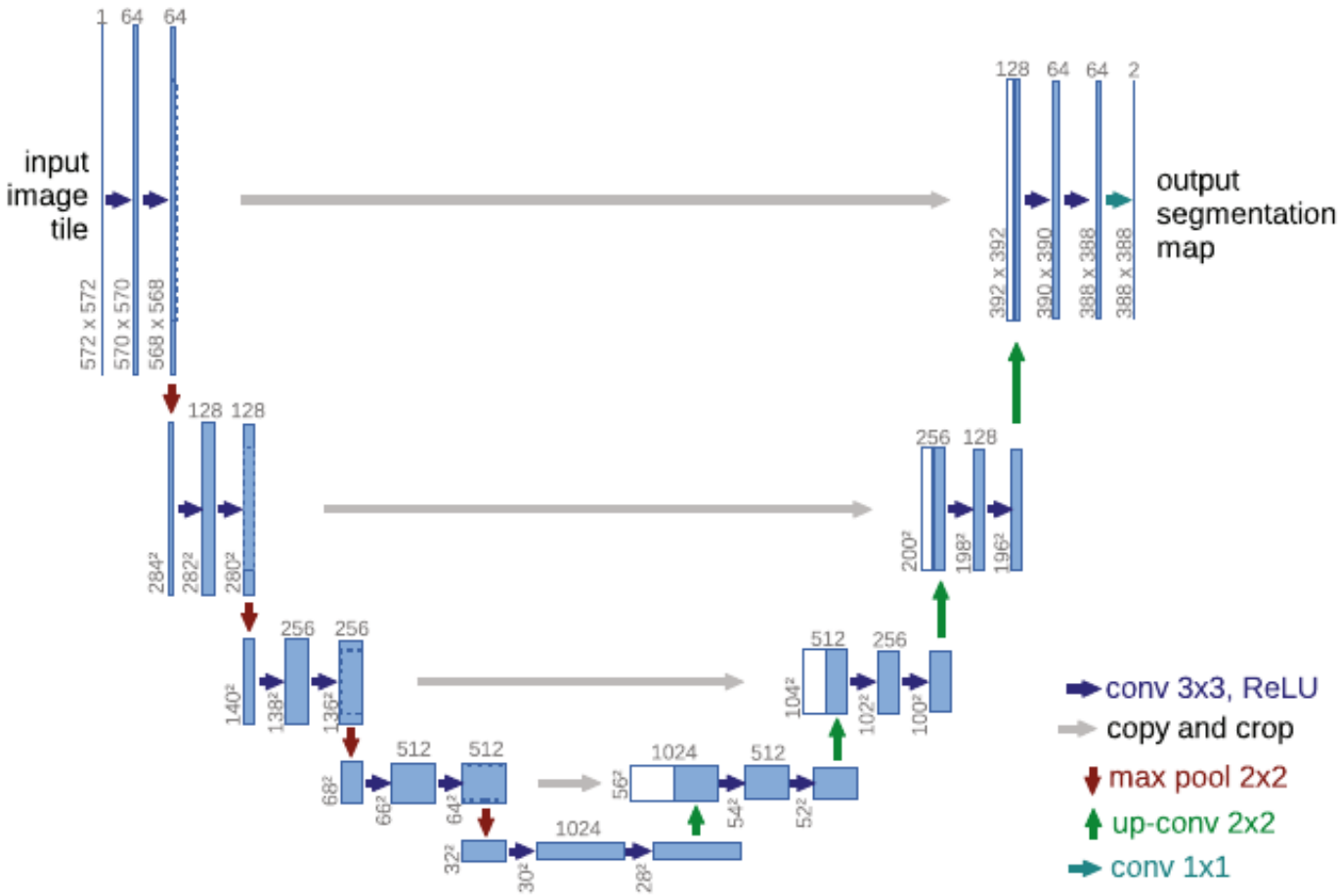
Desafio ISBI (Int. Symp. Biomedical Images) Segmentación automática de estructuras neuronales.

Imágenes de microscopio electrónico.

ISBI seguimiento de celdas 2015. Segmentacion a distintos tiempos.

Objetivo: Requerimos de localización. La etiqueta debe ser asignada a cada pixel de la imagen.

UNETs



La U es una típica estructura de autocodificación. $\mathbf{x}$ es una imagen de 572x572 pixels (bw). La codificamos en un $\mathbf{h}$ de 32x32 pixels (base de la U). Capas residuales (skip connections) mezcladas con down-sampling y up-sampling.

UNETs

- Data augmentation: deformaciones elásticas a las imágenes.
- Función de pérdida: Cross entropy

$$J = \sum_{\mathbf{x}} w(\mathbf{x}) \log(p_{l(\mathbf{x})}(\mathbf{x}))$$

donde l es la etiqueta target de cada pixel.

- Usan etiquetas de los límites de las celdas. Ver contornos en la figura.
- Objetos de la misma clase pegados entre ellos. Como distinguirlos? Mucho peso en la funcion de costo a la clase de limites entre celdas.

$$w(\mathbf{x}) = w_c(\mathbf{x}) + w_0 \exp\left(-\frac{d_1^2 + d_2^2}{2\sigma^2}\right)$$

donde $w_c(\mathbf{x})$ son los pesos para balancear la frecuencia de los pixels de distintas clases. $d_1(\mathbf{x})$ y $d_2(\mathbf{x})$ son la primera y la segunda distancia del borde a la celda mas cercana y a la 2da mas cercana.

Modelos generativos

- No tenemos targets. Aprendizaje no-supervisado.
- Queremos aprender la función densidad de probabilidad.

Esto va mucho mas alla del problema de “estimacion de funcion/al”
Queremos caracterizar a todo el conjunto de los datos con una densidad de probabilidad.
Generación de muestras sintéticas (no existen en los datos).

Aplicaciones:

- Generación de caras “nuevas”.
- Generación de textos (escribir un poema).
- Generación de una voz sintetica.
- Tomadores de decisiones.

Modelos generativos

Métodos clásicos para estimar una PDF son el histograma y kernel density estimation (KDE).

- Curso de la dimensionalidad. No sirven para dimensiones mayores a 10.
- No existe una metología eficiente para generar muestras de distribuciones complejas. (E.g. MCMC, importance sampling).

Alternativa: Aprender una transformación de los datos a un espacio latente de baja dimensionalidad.
Que los datos se representen con distribuciones sencillas (e.g. la normal).
Si tengo el mapa desde el espacio latente al espacio observacional, entonces dada una muestra en el espacio latente puedo encontrar una muestra en el espacio observacional.

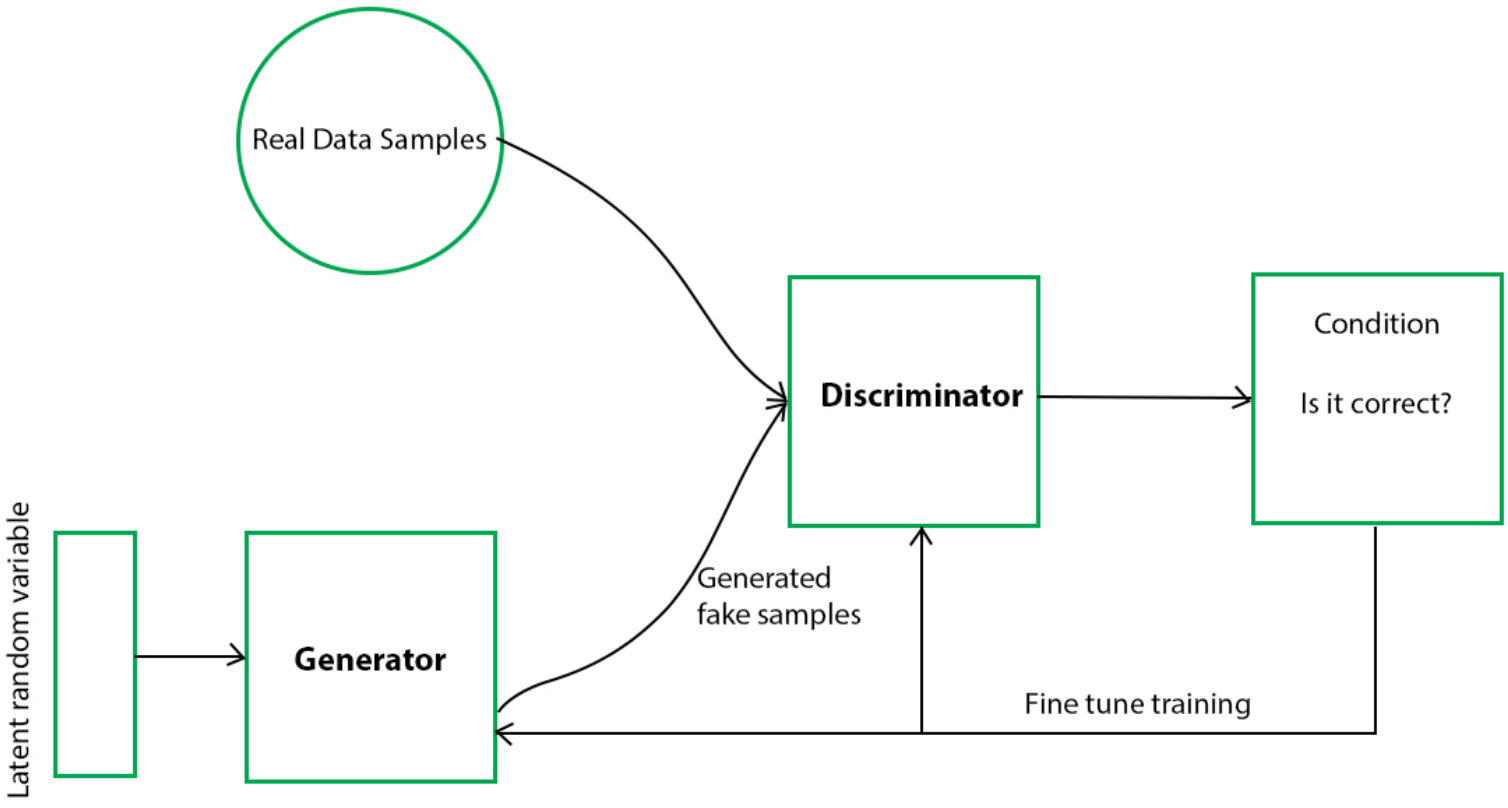
GANs

Generative adversarial Networks. Técnica para generar muestras que son indistinguibles de los datos.
No nos genera una distribución de probabilidad de los datos por lo que no podemos evaluar si un dato pertenece o no al conjunto.

- Para generar una muestra sintética, $\mathbf{x}^*$, se muestrea a partir de una distribución normal en el espacio latente $\mathbf{z}^*$, a esta la transformamos con una red al espacio observacional $\mathbf{x}^* = g(\mathbf{z}^*, \boldsymbol{\theta})$. Esta red es denominada generador.
- Una segunda red $d(\mathbf{x}^*, \boldsymbol{\phi})$ que actua como discriminador, intenta clasificar si la muestra es una real o es ficticia (no pertenece a los datos).

Policía y ladrón.

Estructura de la GAN



Función de pérdida de las GANs

Para el discriminador, como es un clasificador vamos a poner $y_i = 1$ si pensamos que la muestra es real, y $y_i = 0$ si clasifica a la muestra como ficticia.

$$J(\phi) = -\sum_i (1 - y_i) \log\left(1 - \text{sig}\left(f(\mathbf{x}_i, \phi)\right)\right) + y_i \log\left(\text{sig}\left(f(\mathbf{x}_i, \phi)\right)\right)$$

Si tenemos n_D, n_G muestras reales y ficticias, entonces:

$$J(\phi) = -\sum_j^{n_G} \log\left(1 - \text{sig}\left(f(\mathbf{x}_j^*, \phi)\right)\right) - \sum_i^{n_D} \log\left(\text{sig}\left(f(\mathbf{x}_i, \phi)\right)\right)$$

Un término “pesa” las muestras ficticias y el otro las reales. La idea sería encontrar el ϕ los parámetros de la red que nos dan una J mínima.

Función de pérdida de las GANs

Si las muestras las genero con el generador, $\mathbf{x}_j^* = g(\mathbf{z}_j, \boldsymbol{\theta})$,

$$J(\boldsymbol{\theta}, \boldsymbol{\phi}) = -\sum_{j}^{n_G} \log\left(1 - \text{sig}\left(f(g(\mathbf{z}_j, \boldsymbol{\theta}), \boldsymbol{\phi})\right)\right) - \sum_{i}^{n_D} \log\left(\text{sig}\left(f(\mathbf{x}_i, \boldsymbol{\phi})\right)\right)$$

Pero ahora necesito maximizar la J para "engañar" al discriminador que se piense que son muestras reales.

Es decir que para encontrar los $\boldsymbol{\theta}$ requiero maximizar la J o minimizar la $-J$.

En el caso de θ solo depende de un término por lo que tenemos que minimizar es:

$$J_G(\boldsymbol{\theta}) = \sum_{j}^{n_G} \log\left(1 - \text{sig}\left(f(g(\mathbf{z}_j, \boldsymbol{\theta}), \boldsymbol{\phi})\right)\right)$$

Entrenamiento de las GANs

Vamos a tener que entrenar en dos pasos, en una minimizamos la J con respecto a ϕ para que el discriminador pueda distinguir las muestras sintéticas de las reales.

Mientras la maximización de J, o minimización de J_G, con respecto a θ busca que las muestras sintéticas se parezcan reales.

Esto es un minimax game.

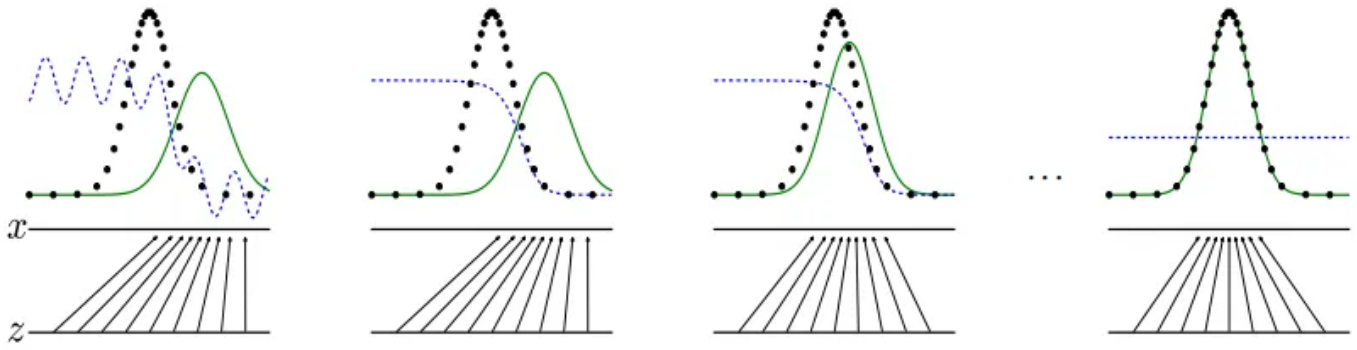
Para cada paso de entrenamieno, generamos n_G muestras latentes $\{z_j, j = 1 : n_G\}$ y las transformamos,$\{x_j^* = g(z_j, \boldsymbol{\theta}), j = 1 : n_G\}$, tomamos n_D muestras de los datos y con estas. Para el descenso de gradientes hacemos primero el paso de J y luego el paso de J_G:

$$\boldsymbol{\phi}_k = \boldsymbol{\phi}_{k-1} - \eta \nabla J(x^*_{1:N_G}, x1 : N_D, \boldsymbol{\phi}_{k-1}, \boldsymbol{\theta}_{k-1})$$

$$\boldsymbol{\theta}_k = \boldsymbol{\theta}_{k-1} - \eta_G \nabla J_G(x^*_{1:N_G}, x1 : N_D, \boldsymbol{\phi}_k, \boldsymbol{\theta}_{k-1})$$

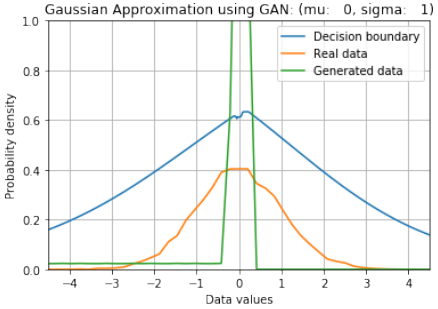
donde N_G, N_D es un minibatch, eg. $N_G = N_D = 64$.

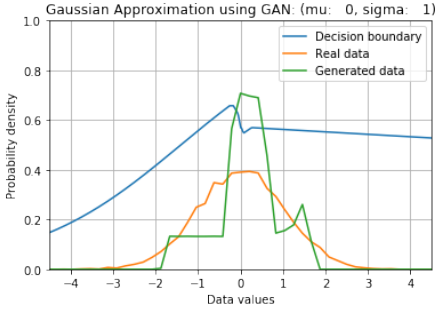
Ejemplo de una Gaussiana

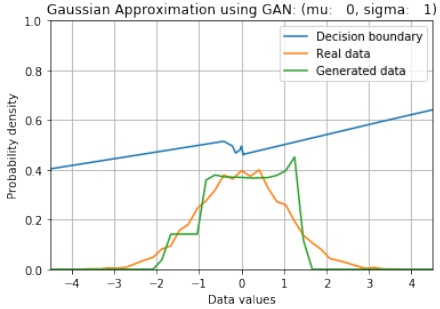


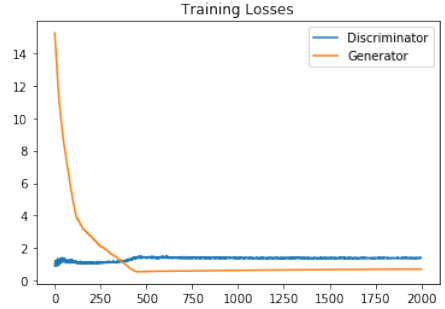
Linea azul a trazos. La distribución discrimitativa. Discrimina muestras de la distribución generativa de los datos p_x (línea a puntos negros) y la generada nuestra (verde).

A partir de puntos de una Gaussiana se transforman a la verde.









Convergencia de las GANs

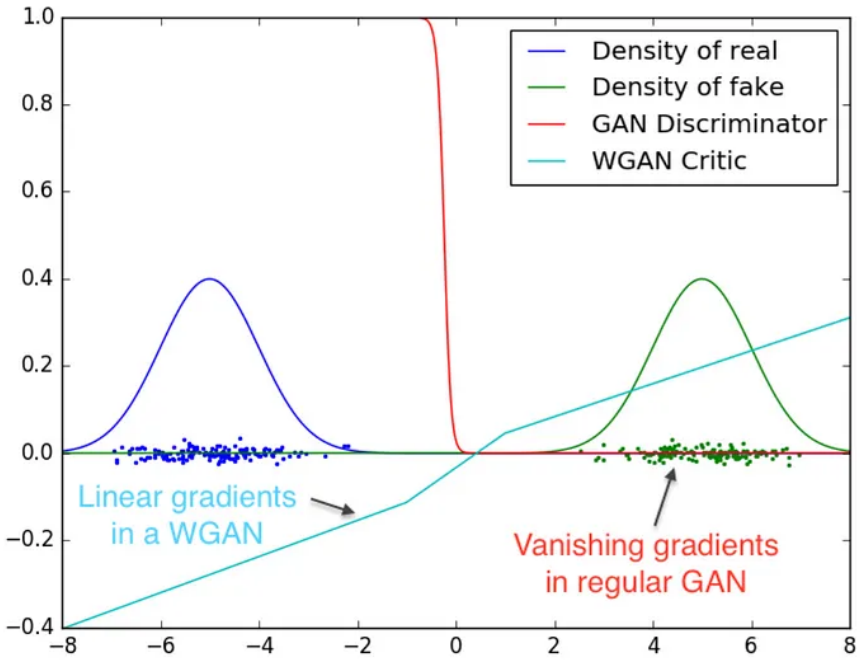
El problema de convergencia de las GANs es un equilibrio de Nash.

- Colapso en la moda.
- Pérdida de modas.
- Inestabilidad y no-convergencia.
- Alta sensibilidad a las métricas e hiperparámetros.

Queremos aprender no solo la media pero tambien el ruido/la incerteza del sistema

Wasserstein GAN

La medida que hay detras de las GANs es la divergencia de Jensen-Shannon se puede usar la métrica de Wasserstein.



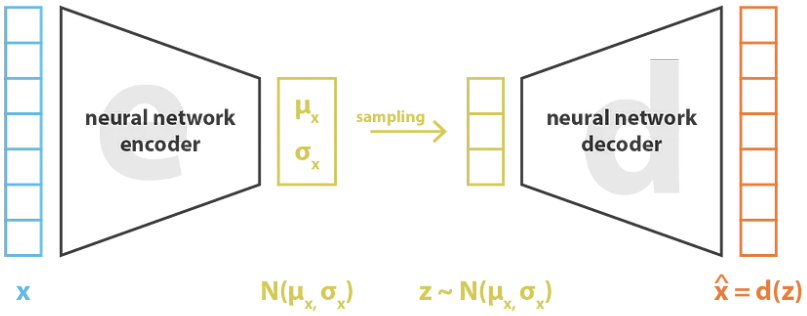
Wasserstein GAN:

- Mejora la estabilidad del aprendizaje.
- Previene el colapso a las modas.
- Permite optimización de los hiperparámetros (por la mejora de la estabilidad).

Arjovsky, M. et al. Wasserstein Generative Adversarial Networks. ICML 2017.

Variational Autoencoders

Los autocodificadores buscan producir replicas de la entrada pasando por un espacio latente de mucho menor dimensionalidad. Ejemplo MNIST.



- Modelo generativo: Queremos determinar la distribución de probabilidad.
- Asumimos que la distribución es Gaussiana.

- Autoencoder que va a tratar de inferir la media y la varianza en el espacio latente.
- Luego debería producir muestras Gaussianas con $\mathbf{z} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$.

Marco probabilístico de un variational autoencoder

Queremos inferir una distribución $p(z|x)$ de probabilidad en el espacio latente.

La reconstrucción $d(z)$ son muestras aleatorias condicionadas a la entrada.

Suponemos la inferencia de un modelo paramétrico, $q_\phi(z|x)$, el codificador.

La red del codificador debe predecir los paramétros de la distribución normal,

$$\boldsymbol{\mu}, \boldsymbol{\sigma} = \text{NeuralNet}_\phi(\mathbf{x})$$

$$q_\phi(\mathbf{z}|\mathbf{x}) = \mathcal{N}(\mathbf{z}; \boldsymbol{\mu}, \boldsymbol{\sigma})$$

donde $\boldsymbol{\sigma}$ es la diagonal de la covarianza $\boldsymbol{\Sigma}$.

$$p(\mathbf{z}|\mathbf{x}) \approx q_\phi(\mathbf{z}|\mathbf{x})$$

ELBO

La inferencia variacional se encarga de optimizar los paramétros de la distribución. La log-verosimilitud de los datos es:

$$\log p_\theta(\mathbf{x}) = \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left(\log \frac{p_\theta(\mathbf{x},\mathbf{z})}{q_\phi(\mathbf{z}|\mathbf{x})}\right) + \mathbb{E}_{q_\phi(\mathbf{z}|\mathbf{x})}\left(\log \frac{q_\phi(\mathbf{z}|\mathbf{x})}{p_\theta(\mathbf{z}|\mathbf{x})}\right)$$

$$\log p_\theta(\mathbf{x}) = \mathcal{L}_{\theta,\phi}(\mathbf{x}) + \mathcal{D}_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})\right)$$

Notando que $\mathcal{D}_{KL}\left(q_\phi(\mathbf{z}|\mathbf{x})||p_\theta(\mathbf{z}|\mathbf{x})\right) \geq 0$,

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) \geq \log p_\theta(\mathbf{x})$$

La funcion de perdida utilizada es la ELBO (evidence lower bound). Si maximizamos la ELBO con respecto a θ, ϕ estamos:

- maximizando la verosimilitud $p_\theta(\mathbf{x})$
- minimizando la KLD, i.e. $q_\phi(\mathbf{z}|\mathbf{x})$ se aproxima a $p_\theta(\mathbf{z}|\mathbf{x})$.

El truco de la reparametrización

En la practica la ELBO la estimamos con una muestra de Monte Carlo,

$$\mathcal{L}_{\theta,\phi}(\mathbf{x}) = \log p_\theta(x_i|z) + \log p(z) - \log q_\phi(z|x_i)$$

Si realizamos muestras de $\boldsymbol{z} = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$, no podemos volver desde $\mathbf{z}$ a θ, en términos del gradiente ya que las z son muestras.
Necesitamos independizar la parte aleatoria de la parte de los parámetros.
Hacemos una transformación:

$$z = \mu + \sigma \odot \epsilon$$

donde $\epsilon = \mathcal{N}(0, 1)$, muestra de una normal estandard.

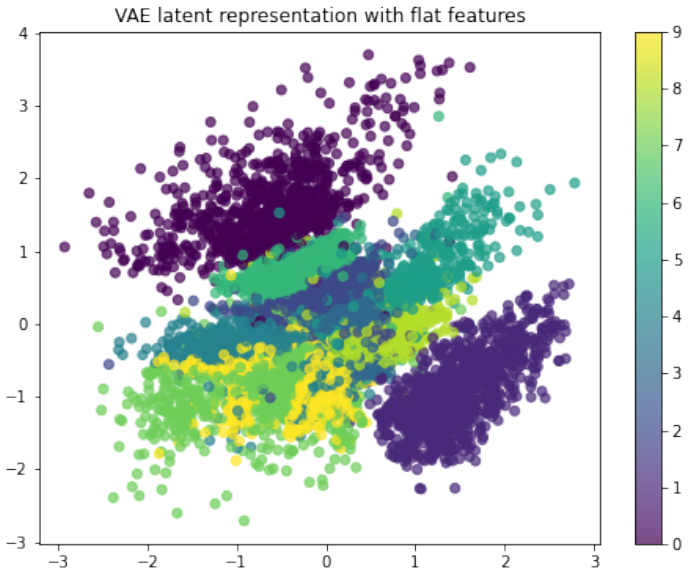
Entonces estamos obligando que el espacio latente tenga media 0 y varianza 1, mientras μ y σ pasan a ser parte de los parámetros del codificador y el decodificador.

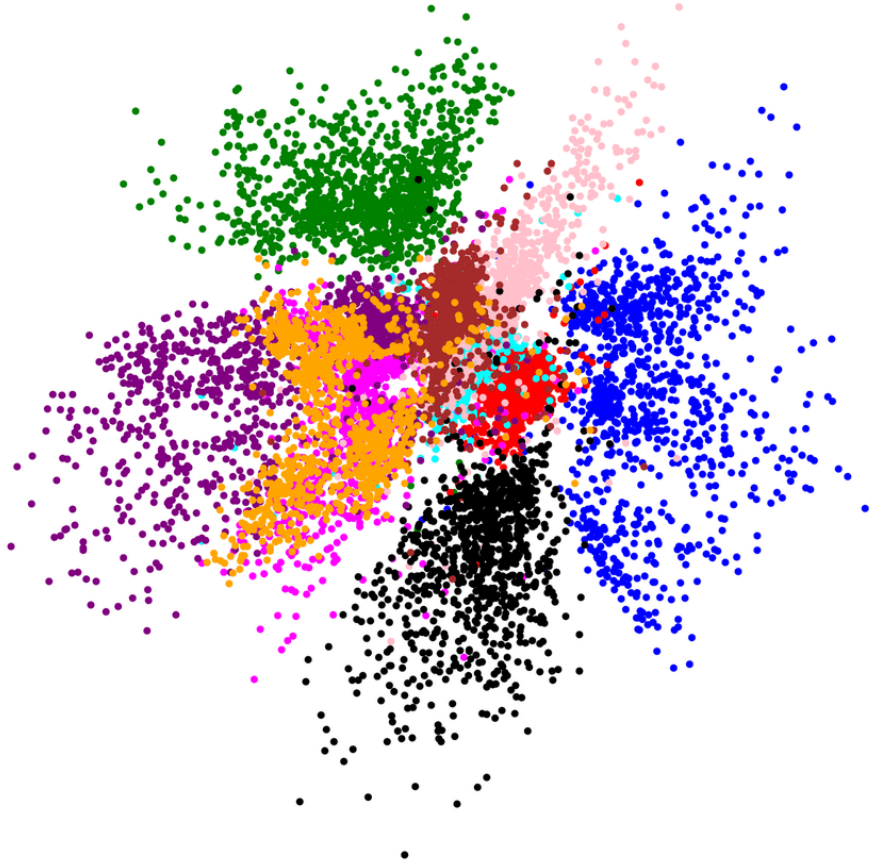
VAE en pytorch

```
class VAE(nn.Module):
    def __init__(self, shape, nhid = 16):
        super(VAE, self).__init__()
        self.dim = nhid
        self.encoder = Encoder(shape, nhid)
        self.decoder = Decoder(shape, nhid)

    def sampling(self, mean, logvar):
        eps = torch.randn(mean.shape).to(device)
        sigma = 0.5 * torch.exp(logvar)
        return mean + eps * sigma

    def forward(self, x):
        mean, logvar = self.encoder(x)
        z = self.sampling(mean, logvar)
        return self.decoder(z), mean, logvar

    def generate(self, batch_size = None):
        z = torch.randn((batch_size, self.dim)).to(device)
        return self.decoder(z)
```

```
def loss_function(x, x_hat, mean, log_var):
    reproduction_loss = nn.functional.binary_cross_entropy(x_hat, x, reduction='sum')
    KLD = - 0.5 * torch.sum(1+ log_var - mean.pow(2) - log_var.exp())
    return reproduction_loss + KLD
```

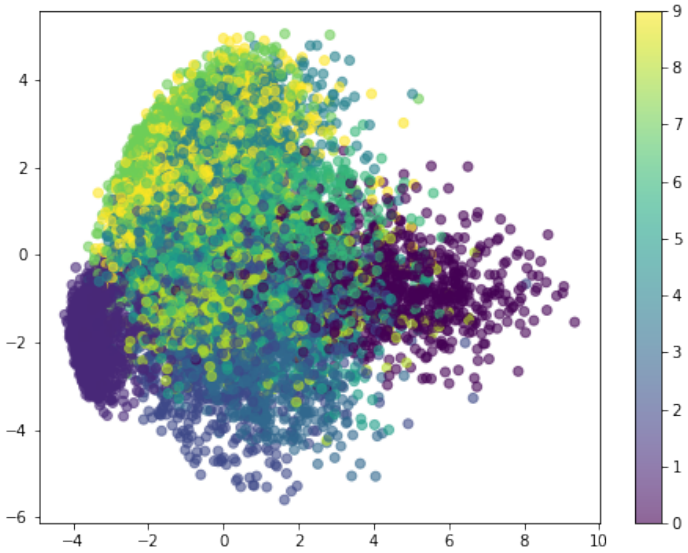
Representación en el espacio latente

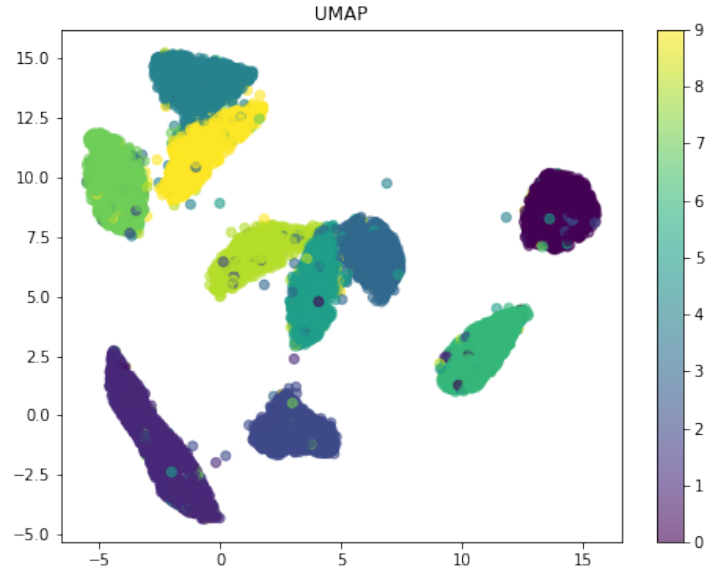


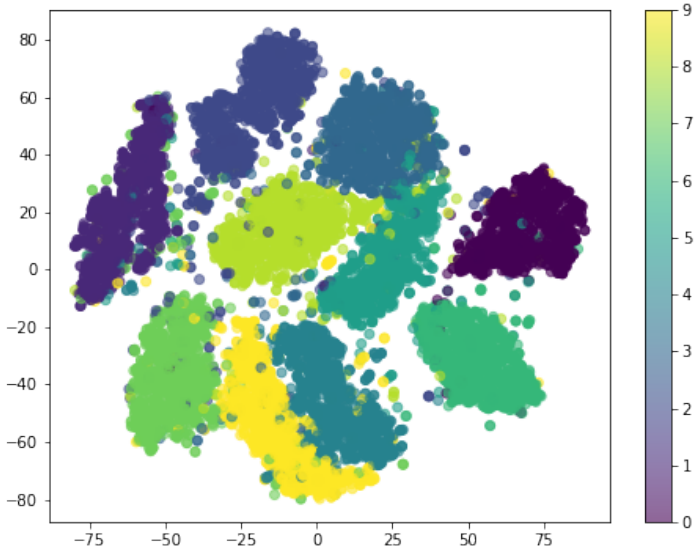


Esto podría pensarse como una reducción de la dimensionalidad de imágenes de 48x48 a 2 dimensiones.

Reducción de la dimensionalidad en MNIST.

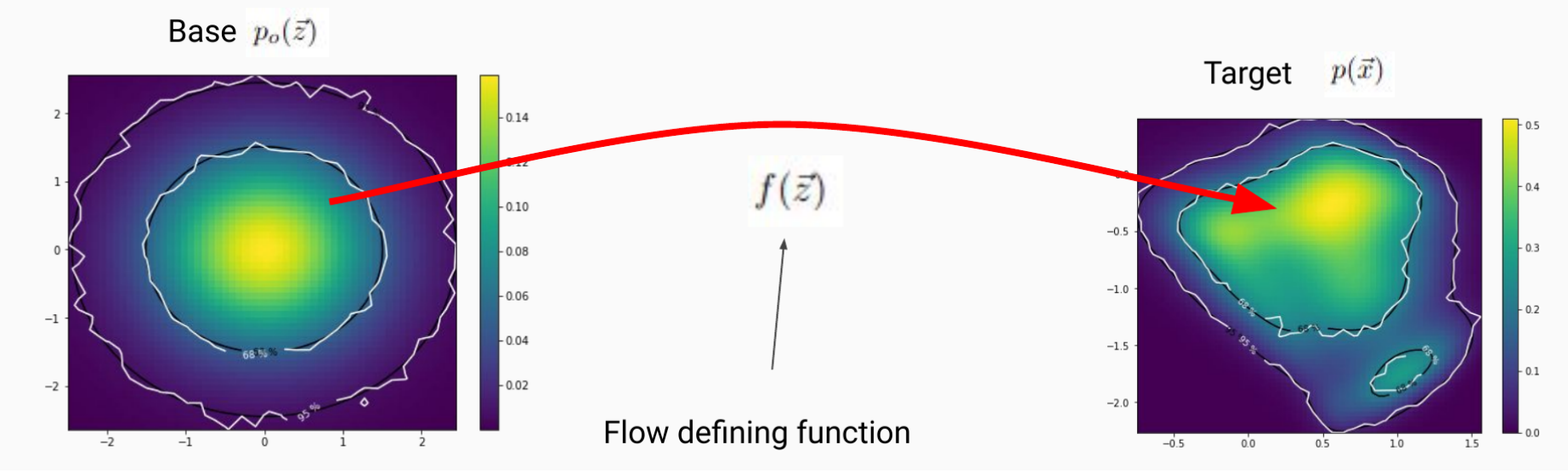






PCA, UMAP y TNSE

Flujos normalizantes



Transformación de una distribución simple (normal) a la distribución de los datos

Rezende D, Mohamed S. Variational inference with normalizing flows. ICML 2015.

Esteban Takak (Courant Institute).

Cristina Turner (FaMAF, UNC).

Flujos normalizantes

- Trabaja con distribuciones arbitrariamente complejas.
- Funciona en alta dimensionalidad y sobre manifolds.
- Permite evaluar analiticamente la probabilidad.
- Genera muestras diferenciables y por lo tanto las esperanzas.

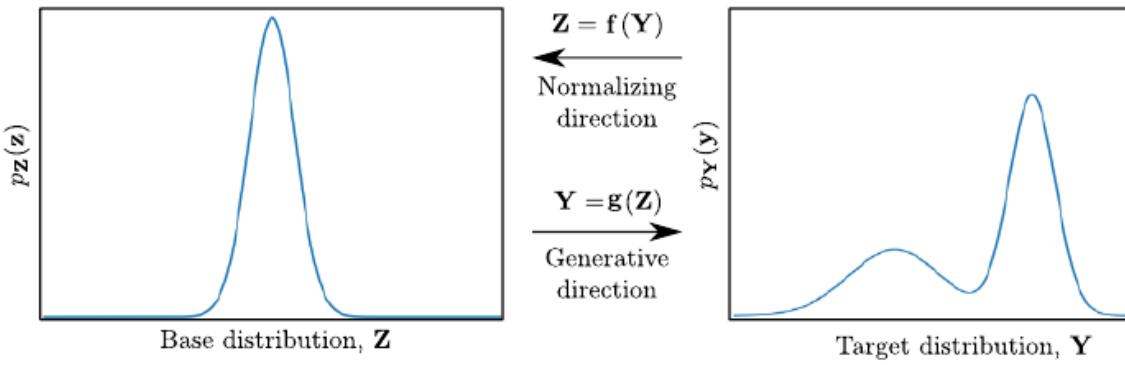
Tabak, E.G. and Turner, C.V., 2013. A family of nonparametric density estimation algorithms. CPAM.

Pulido M., and P. J. vanLeewen, 2019: Sequential Monte Carlo with kernel embedded mappings: The mapping particle filter. JCP.

Flujos invertibles

Proponemos una transformación de variables $x = f_\phi(z)$.
Produciendo una transformación de densidades:

$$p_\phi(x) = \left| \det \frac{\partial f_\phi(z)}{\partial z} \right|^{-1} p(z)$$



Si tenemos esta transformación luego $z^* \sim p(z) \quad \rightarrow \quad x^* = f_\phi(z^*)$
Requerimos la inversa para evaluar la verosimilitud $z = f_\phi^{-1}(x)$.
Función biyectiva para que exista la inversa + Función diferenciable $\rightarrow$ Diffeomorfismo.

$$p(x) = p_0(f^{-1}(x) \left| \frac{\det \partial f_\phi^{-1}(x)}{\partial x} \right|$$

Entrenamiento

Dada las muestras el negativo de la log-likelihood:

$$nll(x) = \sum_{n=1}^{N} \left[\log \left| \det \frac{\partial f_\phi(z_n)}{\partial z} \right| - \log p(z_n) \right]$$

donde $z_n = f_\phi^{-1}(x_n)$ son la transformación de cada una de las muestras.
En general se asume que $f_\phi(z)$ es una red neuronal con múltiples capas k, luego

$$x = f_K(f_{K-1}(\cdots f_1(z)\,)$$

$$z = f_\phi^{-1}(x) = f_1^{-1}(f_2^{-1}(\cdots f_K^{-1}(x)\cdots)$$

$$\left| \frac{\partial f_\phi(z_n)}{\partial z} \right| = \left| \frac{\partial f_k}{\partial f_{k-1}} \right| \cdot \left| \frac{\partial f_{k-1}}{\partial f_{k-2}} \cdots \left| \frac{\partial f_1}{\partial z} \right| \right|$$

Potenciales capas de la red:

- Flujos lineales
- Flujos por componentes nolineales.
- Flujos autoregresivos $h_d = g[h_d, \phi(h_{1:d-1})]$
- Flujos residuales

Aplicación de flujos normalizantes

Generación de imágenes sintéticas