

Modelos Generativos Profundos para Imágenes

Modelos autorregresivos

Pablo Musé

pmuse@fing.edu.uy

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería



UNIVERSIDAD
DE LA REPÚBLICA
URUGUAY

Agenda

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basado en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

PixelRNN

PixelCNN

Transformers generativos

Aplicaciones

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basados en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

PixelRNN

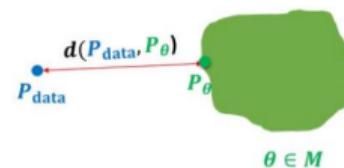
PixelCNN

Transformers generativos

Aplicaciones

Modelado generativo

Recordemos el problema que queremos resolver:



$$\mathbf{x}_n \sim p_{\text{data}}(\mathbf{x}), n = 1, \dots, N \quad \text{Familia de modelos}$$

- **Objetivo:** aprender una densidad $p(\mathbf{x})$ a partir de $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ que nos permita:
 - Generar muestras realistas
 - Tener una estimación fiable de la densidad (e.g. para detección de anomalías)
 - Aprender buenas representaciones (**vectores de features**) de los datos
- **¿Cómo representar $p(\mathbf{x})$?**
- **¿Cómo aprender o estimar $p(\mathbf{x})$?**

Redes bayesianas vs. modelos neuronales: modelos autorregresivos

$\mathbf{x} = (x_1, x_2, \dots, x_d) \sim p_{data}$ una muestra (e.g. imagen de d pixels).

- **Modelo general:**

$$p(x_1, \dots, x_d) = p(x_1) p(x_2 | x_1) p(x_3 | x_1, x_2) \dots p(x_d | x_1, \dots, x_{d-1}).$$

- **Redes bayesianas:** imponemos estructura asumiendo independencia condicional:

$$p(x_1, \dots, x_d) \approx p(x_1) p(x_2 | x_1) p(x_3 | \cancel{x_1}, x_2) \dots p(x_d | \cancel{x_1}, \dots, \cancel{x_{d-2}}, x_{d-1}).$$

- **Modelos Neuronales:**

$$p(x_1, \dots, x_d) \approx p(x_1) p(x_2 | x_1) p_{\text{NeuralNet}}(x_3 | x_1, x_2) \dots p_{\text{NeuralNet}}(x_d | x_1, \dots, x_{d-1}).$$

⇒ Asumimos una forma funcional específica, paramétrica, para las densidades condicionales: **redes neuronales profundas (pueden aproximar cualquier función)**.

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basados en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

PixelRNN

PixelCNN

Transformers generativos

Aplicaciones

Ejemplo: modelo generativo para MNIST

- \mathcal{D} conjunto de dígitos escritos a mano (MNIST binarizado)



- Cada imagen tiene $d = 28 \times 28 = 784$ pixels. Cada píxel puede ser negro (0) o blanco (1).
- **Objetivo:** aprender una distribución de probabilidad $p(\mathbf{x}) = p(x_1, \dots, x_{784})$ sobre $\mathbf{x} \in \{0, 1\}^{784}$ tal que cuando $\mathbf{x} \sim p(\mathbf{x})$, \mathbf{x} se parezca a un dígito.
- **Proceso en dos etapas:**
 - ① Definir una familia paramétrica de modelos $\{p_\theta(\mathbf{x}), \theta \in \Theta\}$
 - ② Estimar el mejor θ posible a partir de \mathcal{D} .

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basados en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

PixelRNN

PixelCNN

Transformers generativos

Aplicaciones

Modelos Neuronales para clasificación

- **Objetivo:** intentar predecir el valor de cada pixel $x_i \in \{0, 1\}$ a partir de los pixels anteriores $\mathbf{x}_{<i} \in \{0, 1\}^{i-1}$ con algún clasificador binario $p(x_i = 1 | \mathbf{x}_{<i}; \alpha_i) = f(\mathbf{x}_{<i}; \alpha_i)$.
- Algunas opciones simples:

Regresión logística: sea $z(\alpha, \mathbf{x}_{<i}) = \alpha_{i,0} + \sum_{j=1}^{i-1} \alpha_{i,j} x_j$,

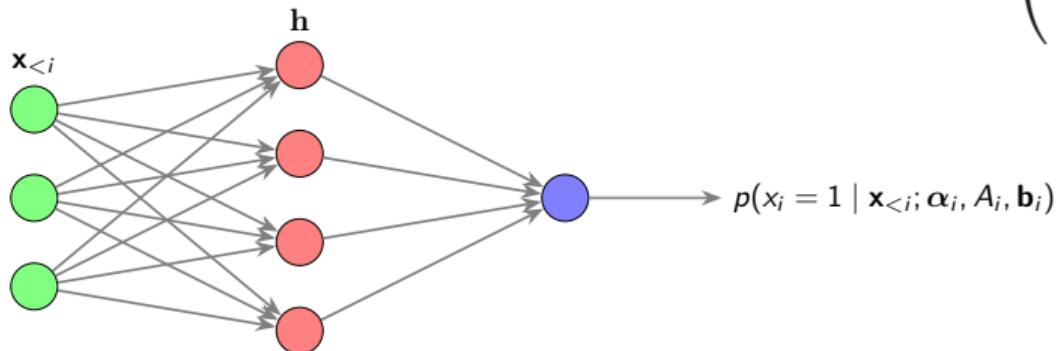
$$p_{logit}(x_i = 1 | \mathbf{x}_{<i}; \alpha_i) = \sigma(z(\alpha_i, \mathbf{x}_{<i})), \quad \text{con } \sigma(z) = 1/(1 + e^{-z}) \text{ (sigmoide)}.$$

Modelos Neuronales para clasificación

- **Objetivo:** intentar predecir el valor de cada pixel $x_i \in \{0, 1\}$ a partir de los pixels anteriores $\mathbf{x}_{<i} \in \{0, 1\}^{i-1}$ con algún clasificador binario $p(x_i = 1 | \mathbf{x}_{<i}; \alpha_i) = f(\mathbf{x}_{<i}; \alpha_i)$.
- Algunas opciones simples:

Perceptrón multicapa: $\mathbf{h}_i(\mathbf{x}; A_i, \mathbf{b}_i) : \{0, 1\}^{i-1} \rightarrow \mathbb{R}^{m_i}$ función no lineal de la de entrada,

$$p_{MLP}(x_i = 1 | \mathbf{x}_{<i}; \alpha_i, A_i, \mathbf{b}_i) = \sigma \left(\alpha_{0,i} + \sum_{j=1}^{m_i} \alpha_{i,j} h_j \right)$$



- Más flexible
- Más parámetros:

$$A_i, \mathbf{b}_i, \alpha_i, i = 1, \dots, d$$

MNIST binarizado: modelo autoregresivo para la función logística

- **Elegimos un orden para las V.A.s** X_1, \dots, X_d , e.g. escaneo rasterizado desde el pixel superior izquierdo X_1 hasta el inferior derecho X_{784} .
- Regla de la cadena:

$$p(x_1, \dots, x_{784}) = p(x_1)p(x_2 | x_1)p(x_3 | x_1, x_2) \dots p(x_{784} | x_1, \dots, x_{783}).$$

- Algunas condicionales son demasiado complejas para almacenarse en tablas:

$$\begin{aligned} p(x_1, \dots, x_d) &= \textcolor{orange}{p}_{CPT}(x_1; \alpha_1) \times \textcolor{green}{p}_{logit}(x_2 | x_1; \alpha_2) \\ &\quad \times \textcolor{green}{p}_{logit}(x_3 | x_1, x_2; \alpha_3) \times \dots \times \textcolor{green}{p}_{logit}(x_d | x_1, \dots, x_{d-1}; \alpha_d) \end{aligned}$$

con

$$\textcolor{orange}{p}_{CPT}(x_1 = 1; \alpha_1) = \alpha_1, \quad p(x_1 = 0) = 1 - \alpha_1$$

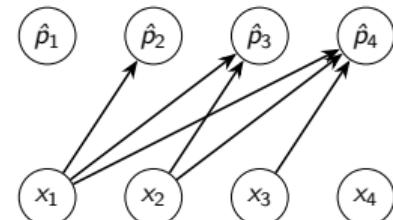
$$\textcolor{green}{p}_{logit}(x_2 = 1 | x_1; \alpha_2) = \sigma(\alpha_{2,0} + \alpha_{2,1}x_1)$$

$$\textcolor{green}{p}_{logit}(x_3 = 1 | x_1, x_2; \alpha_3) = \sigma(\alpha_{3,0} + \alpha_{3,1}x_1 + \alpha_{3,2}x_2)$$

...

Fully Visible Sigmoid Belief Network (FVSBN)

Este modelo que acabamos de definir se llama **Fully Visible Sigmoid Belief Network**



- Las V.A.s condicionales $X_i | X_1, \dots, X_{i-1}$ son Bernoulli con parámetros

$$\hat{p}_i = p(x_i = 1 | \mathbf{x}_{<i}; \boldsymbol{\alpha}_i) = \sigma \left(\alpha_{i,0} + \sum_{j=1}^{i-1} \alpha_{i,j} x_j \right).$$

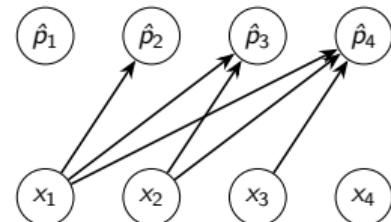
- ¿Cómo evaluar $p(x_1, \dots, x_{784})$?

Multiplicar todas las condicionales (factores). Ejemplo ($d = 4$):

$$p(x_1 = 0, x_2 = 1, x_3 = 1, x_4 = 0) = (1 - \hat{p}_1) \times \hat{p}_2 \times \hat{p}_3 \times (1 - \hat{p}_4).$$

Fully Visible Sigmoid Belief Network (FVSBN)

Este modelo que acabamos de definir se llama **Fully Visible Sigmoid Belief Network**



- Las V.A.s condicionales $X_i | X_1, \dots, X_{i-1}$ son Bernoulli con parámetros

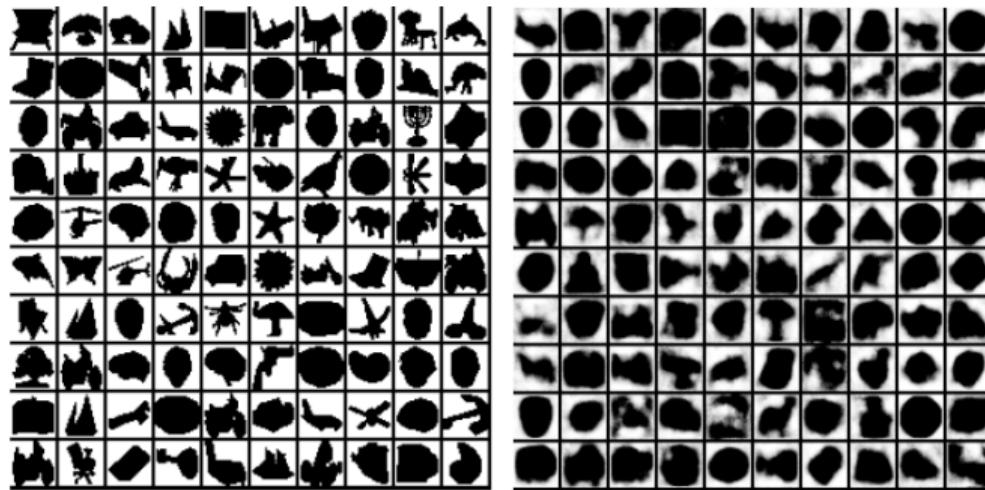
$$\hat{p}_i = p(x_i = 1 | \mathbf{x}_{<i}; \boldsymbol{\alpha}_i) = \sigma \left(\alpha_{i,0} + \sum_{j=1}^{i-1} \alpha_{i,j} x_j \right).$$

- ¿Cómo muestrear de $p(x_1, \dots, x_{784})$?

- ① Muestrear $\bar{x}_1 \sim p(x_1) = \text{Ber}(\hat{p}_1)$
- ② Muestrear $\bar{x}_2 \sim p(x_2 | x_1 = \bar{x}_1) = \text{Ber}(\hat{p}_2)$
- ③ Muestrear $\bar{x}_3 \sim p(x_3 | x_1 = \bar{x}_1, x_2 = \bar{x}_2) = \text{Ber}(\hat{p}_3) \dots$

¿Cuántos parámetros (en los vectores $\boldsymbol{\alpha}^i$)? $1 + 2 + 3 + \dots + d \approx d^2/2$

Resultados de FVSBN



- Muestras de entrenamiento (*Caltech 101 Silhouettes*) (izquierda)
- Muestras generadas con FVSBN (derecha).

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basado en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

PixelRNN

PixelCNN

Transformers generativos

Aplicaciones

NADE: Neural Autoregressive Density Estimation

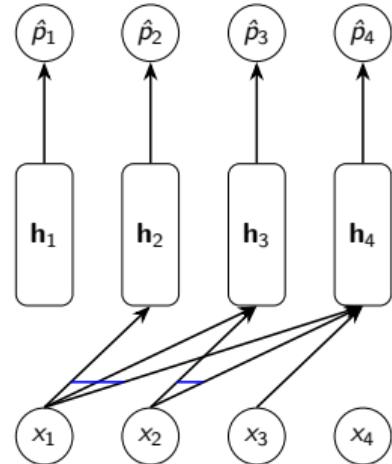
Podemos mejorar FVSBN remplazando la regresión logísitica por una red neuronal de una capa oculta \mathbf{h}_i :

$$\mathbf{h}_i = \sigma(A_i \mathbf{x}_{<i} + \mathbf{c}_i)$$

$$\hat{p}_i = p(x_i | x_1, \dots, x_{i-1}; \underbrace{A_i, \mathbf{c}_i, \boldsymbol{\alpha}_i, b_i}_{\text{parámetros}}) = \sigma(\boldsymbol{\alpha}_i^T \mathbf{h}_i + b_i).$$

- Por ejemplo

$$\mathbf{h}_2 = \sigma \left(\underbrace{\begin{pmatrix} \vdots \\ \vdots \end{pmatrix}}_{A_2} x_1 + \underbrace{\begin{pmatrix} \vdots \\ \vdots \end{pmatrix}}_{\mathbf{c}_2} \right), \quad \mathbf{h}_3 = \sigma \left(\underbrace{\begin{pmatrix} \vdots \\ \vdots \\ \vdots \end{pmatrix}}_{A_3} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \underbrace{\begin{pmatrix} \vdots \\ \vdots \end{pmatrix}}_{\mathbf{c}_3} \right)$$

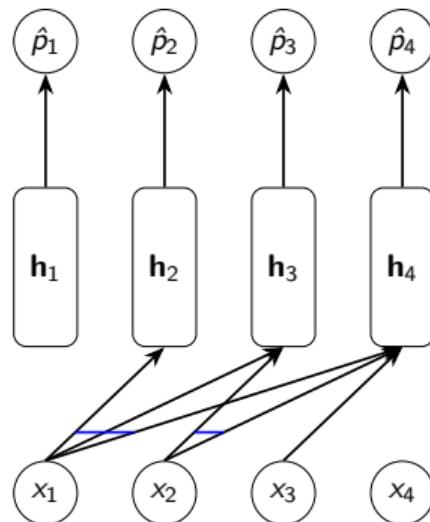


NADE: Neural Autoregressive Density Estimation

También **se atan los pesos** para reducir parámetros
y acelerar cálculo (líneas azules en figura):

$$\mathbf{h}_i = \sigma(\mathbf{W}_{\cdot, <i} \mathbf{x}_{<i} + \mathbf{c})$$

$$\begin{aligned}\hat{p}_i &= p(x_i | x_1, \dots, x_{i-1}; \mathbf{W}_{\cdot, <i}, \mathbf{c}, \boldsymbol{\alpha}_i, b_i) \\ &= \sigma(\boldsymbol{\alpha}_i^T \mathbf{h}_i + b_i).\end{aligned}$$



NADE: Neural Autoregressive Density Estimation

$$\mathbf{h}_i = \sigma(W_{.,<i} \mathbf{x}_{<i} + \mathbf{c})$$

$$\hat{p}_i = p(x_i \mid x_1, \dots, x_{i-1}; W_{.,<i}, \mathbf{c}, \alpha_i, b_i) = \sigma(\alpha_i \mathbf{h}_i + b_i).$$

- Por ejemplo

$$\mathbf{h}_2 = \sigma \left(\underbrace{\begin{pmatrix} \vdots \\ w_1 \\ \vdots \end{pmatrix}}_{W_{.,<2}} x_1 + \mathbf{c} \right), \quad \mathbf{h}_3 = \sigma \left(\underbrace{\begin{pmatrix} \vdots & \vdots \\ w_1 & w_2 \\ \vdots & \vdots \end{pmatrix}}_{W_{.,<3}} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} + \mathbf{c} \right), \quad \mathbf{h}_4 = \sigma \left(\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ w_1 & w_2 & w_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{W_{.,<4}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \mathbf{c} \right)$$

NADE: Neural Autoregressive Density Estimation

$$\mathbf{h}_i = \sigma(W_{.,<i} \mathbf{x}_{<i} + \mathbf{c})$$

$$\hat{p}_i = p(x_i | x_1, \dots, x_{i-1}; W_{.,<i}, \mathbf{c}, \alpha_i, b_i) = \sigma(\alpha_i \mathbf{h}_i + b_i).$$

$$\mathbf{h}_4 = \sigma \left(\underbrace{\begin{pmatrix} \vdots & \vdots & \vdots \\ w_1 & w_2 & w_3 \\ \vdots & \vdots & \vdots \end{pmatrix}}_{W_{.,<4}} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \mathbf{c} \right)$$

Si $\mathbf{h}_i \in \mathbb{R}^h$, ¿cuántos parámetros totales?

Lineal en la dimensión d : pesos $W \in \mathbb{R}^{h \times (d-1)}$, sesgos $\mathbf{c} \in \mathbb{R}^h$, y d vectores de coeficientes de regresión logística $\alpha_i, b_i \in \mathbb{R}^{h+1}$. La probabilidad se evalúa en $O(dh)$.

Resultados de NADE



- Muestras de NADE entrenado en MNIST (izquierda)
- Probabilidades condicionales \hat{p}_i (derecha).

Distribuciones discretas generales

- ¿Cómo modelar variables aleatorias discretas no binarias $X_i \in \{1, \dots, K\}$? E.g. intensidades de píxeles que varían de 0 a 255.
- Sea $\hat{\mathbf{p}}_i$ una parametrización de una distribución categórica

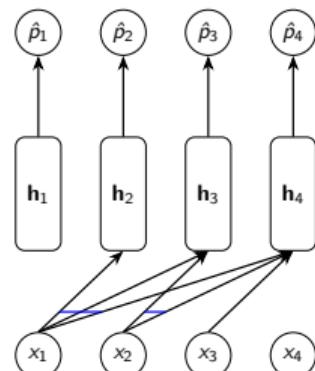
$$\mathbf{h}_i = \sigma(W_{:, < i} \mathbf{x}_{< i} + \mathbf{c})$$

$$p(x_i | x_1, \dots, x_{i-1}) = \text{Cat}\left(p_i^1, \dots, p_i^K\right)$$

$$\hat{\mathbf{p}}_i = \left(p_i^1, \dots, p_i^K\right) = \text{softmax}(A_i \mathbf{h}_i + \mathbf{b}_i).$$

- Softmax generaliza la función sigmoide/logística $\sigma(\cdot)$ y transforma un vector de K números en un vector de K probabilidades (no negativas, suman 1):

$$\text{softmax}(\mathbf{a}) = \text{softmax}\left(a^1, \dots, a^K\right) = \left(\frac{\exp(a^1)}{\sum_i \exp(a^i)}, \dots, \frac{\exp(a^K)}{\sum_i \exp(a^i)}\right)$$



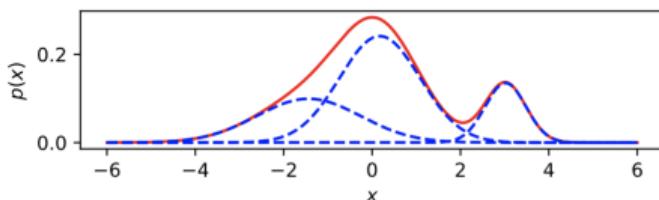
Distribuciones discretas generales

- **¿Cómo se entrena?** $\mathcal{D} = \{\mathbf{x}_1, \dots, \mathbf{x}_N\}$ i.i.d., conjunto de entrenamiento.

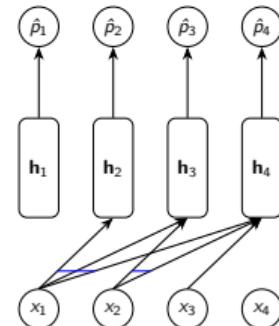
$$\begin{aligned}\text{Maximizando } \log p(\mathcal{D}) &= \log \prod_{n=1}^N p(\mathbf{x}_n) = \sum_{n=1}^N \log p(\mathbf{x}_n) \\&= \sum_{n=1}^N \log \prod_{i=1}^d p(x_{n,i} \mid \mathbf{x}_{n,<i}) \quad (\mathbf{x}_{<1} = \emptyset) \\&= \sum_{n=1}^N \sum_{i=1}^d \log p(x_{n,i} \mid \mathbf{x}_{n,<i}) \\&= \sum_{n=1}^N \sum_{i=1}^d \log \text{Cat}(x_{n,i}; p_{n,i}^1, p_{n,i}^2, \dots, p_{n,i}^K) \\&= \sum_{n=1}^N \sum_{i=1}^d \sum_{k=1}^K \mathbb{1}_{[x_{n,i}=k]} \log p_{n,i}^k = \sum_{n=1}^N \sum_{i=1}^d \log p_{n,i}^{(x_{n,i})}.\end{aligned}$$

RNADE: Real Neural Autoregressive Density Estimation

- ¿Cómo modelar V.A.s continuas $X_i \in \mathbb{R}$? (e.g. audio)
- **Solución:** hacer que $\hat{\mathbf{p}}_i$ parametrize una distribución continua
- Ejemplo: mezcla uniforme de K gaussianas.



$$p(x_i | x_1, \dots, x_{i-1}) = \sum_{j=1}^K \frac{1}{K} \mathcal{N}(x_i; \mu_i^j, \sigma_i^j)$$
$$\mathbf{h}_i = \sigma(W_{:, < i} \mathbf{x}_{< i} + \mathbf{c})$$
$$\hat{\mathbf{p}}_i = (\mu_i^1, \dots, \mu_i^K, \sigma_i^1, \dots, \sigma_i^K) = f(\mathbf{h}_i)$$



$\hat{\mathbf{p}}_i$ define la media y la desviación estándar de cada una de las K gaussianas (μ_i^j, σ_i^j) . Se puede usar $\log \sigma_i$ en lugar de σ_i , con $\exp(\cdot)$, para asegurar no negatividad.

- **¿Cómo se entrena?**

Maximizando $\log p(\mathcal{D}) = \sum_{n=1}^N \log p(\mathbf{x}_n)$ (**Ejercicio:** similar al caso de los modelos discretos, pero usando la optimización del modelo MoG vista en clase anterior)

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basados en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

PixelRNN

PixelCNN

Transformers generativos

Aplicaciones

Modelos autorregresivos mediante autoencoders

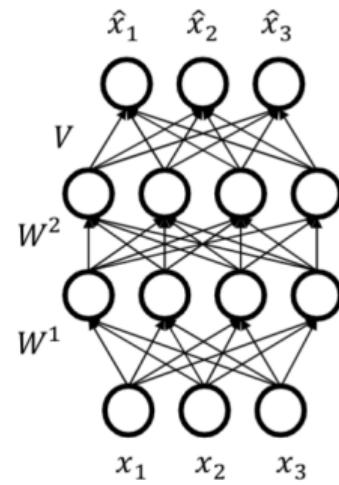
FVSBN y NADE se parecen bastante a un *autoencoder*.

- **¿Qué es un autoencoder?**

- Un *encoder* $E(\cdot)$, e.g., $E(\mathbf{x}) = \sigma(W^2(W^1\mathbf{x} + \mathbf{b}^1) + \mathbf{b}^2)$.
- Un *decoder* tal que $D(E(\mathbf{x})) \approx \mathbf{x}$, e.g., $D(\mathbf{h}) = \sigma(V\mathbf{h} + \mathbf{c})$.
- Función de costo para el dataset \mathcal{D} (e.g. V.A. continua):

$$\min_{W^1, W^2, b^1, b^2, V, c} \sum_{x \in \mathcal{D}} \sum_i (x_i - \hat{x}_i)^2.$$

- E y D están restringidos para que no aprendan la identidad.
- $E(\mathbf{x})$ debería ser una buena representación comprimida de \mathbf{x} (aprendizaje de características)



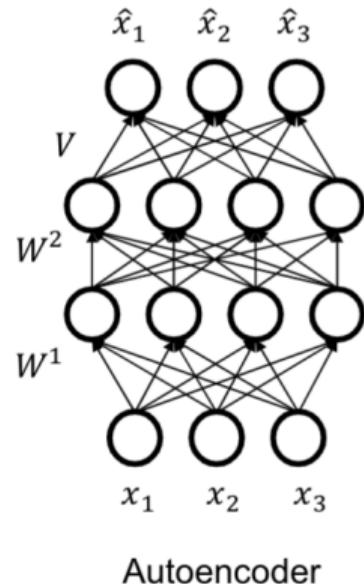
Autoencoder

Modelos autorregresivos vs. autoencoders

- Un autoencoder **básico** no es un modelo generativo: no define una distribución sobre \mathbf{x} de la cual muestrear para generar nuevas muestras.
- ¿Podemos obtener un modelo generativo a partir de un autoencoder?

Sí, pero tiene que corresponder a red bayesiana válida (DAG):

⇒ Necesitamos un orden para la regla de la cadena.

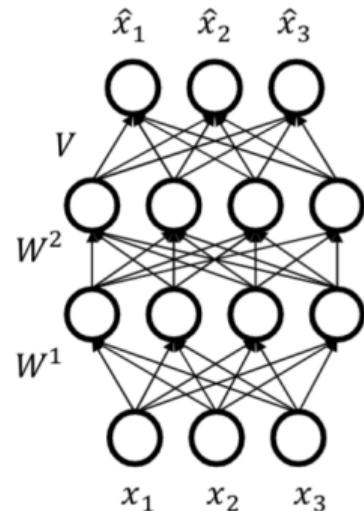


Autoencoder

Modelos autorregresivos vs. autoencoders

Ejemplo: si el orden es 1, 2, 3, entonces

- \hat{x}_1 no puede depender de ninguna entrada
 $x = (x_1, x_2, x_3)$. En tiempo de generación, no necesitamos ninguna entrada para comenzar.
- \hat{x}_2 solo puede depender de x_1 .
- \hat{x}_3 solo puede depender de x_1 y x_2 .

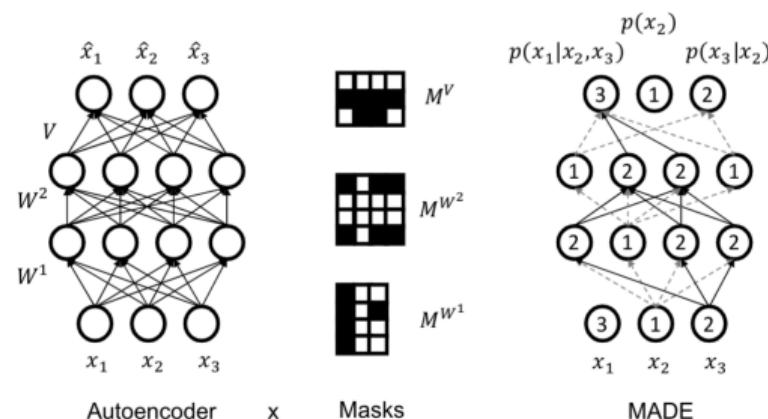


Autoencoder

Bonus: una sola red neuronal (con d entradas y salidas) para producir todos los parámetros en una sola pasada (a diferencia de NADE que requiere d pasadas). Mucho más eficiente en hardware moderno.

MADE: Masked Autoencoder for Distribution Estimation

- **Desafío:** construir un autoencoder que sea autorregresivo (estructura DAG)
- **Solución:** usar **máscaras** para prohibir ciertos caminos.

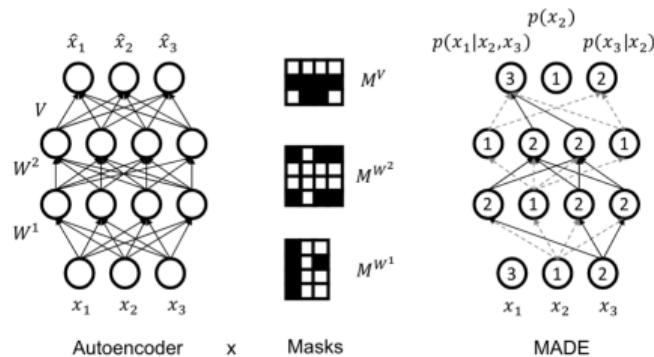


Ejemplo: Si el orden fuera X_2, X_3, X_1 ,

$$p(x_1, x_2, x_3) = p(x_2)p(x_3 | x_2)p(x_1 | x_2, x_3).$$

- ⇒ La neurona que produce $\hat{x}_2 = p(x_2)$ no puede depender de ninguna entrada.
- ⇒ La neurona que produce $p(x_3 | x_2)$ solo depende de x_2 .
- ⇒ La neurona que produce $p(x_1 | x_2, x_3)$ solo depende de x_2 y x_3 .

MADE: Masked Autoencoder for Distribution Estimation



Algoritmo de enmascarado: para cada neurona en una capa oculta:

- ① Elegir un entero aleatorio $i \in [1, d - 1]$.
- ② Esa unidad solo puede depender de las primeras i entradas (según el orden elegido).
- ③ Agregar máscara para preservar este invariante: conectar a todas las unidades en la capa anterior con un número asignado menor o igual (menor estricto en la última capa).

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basados en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

PixelRNN

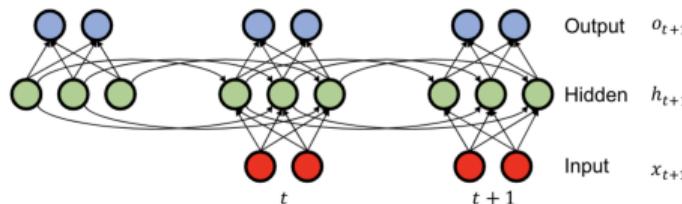
PixelCNN

Transformers generativos

Aplicaciones

RNN: Redes Neuronales Recurrentes

- **Desafío:** modelar $p(x_t | \mathbf{x}_{1:t-1}; \alpha^t)$. El histórico de datos $\mathbf{x}_{1:t-1}$ sigue creciendo.
- **Idea:** mantener un resumen hasta el tiempo t y actualizarlo recursivamente.



$$\mathbf{h}_{t+1} = \tanh(W_{hh}\mathbf{h}_t + W_{xh}\mathbf{x}_{t+1})$$

Ecuación de estado

$$\mathbf{o}_{t+1} = W_{hy}\mathbf{h}_{t+1}$$

Ecuación de medida o predicción

$$\mathbf{h}_0 = \mathbf{b}_0$$

Inicialización.

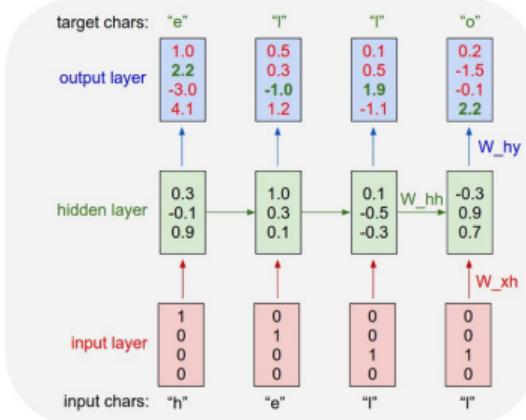
- La capa oculta \mathbf{h}_t es un resumen de las entradas vistas hasta el momento t
- La capa de salida \mathbf{o}_{t-1} especifica los parámetros para la condicional $p(x_t | \mathbf{x}_{1:t-1})$
- Parametrizada por \mathbf{b}_0 (inicialización), y las matrices W_{hh} , W_{xh} , W_{hy} : **número constante de parámetros con respecto a d**

Ejemplo: RNN de Caracteres

- Supongamos que $x_i \in \{h, e, l, o\}$.

- Modelo autorregresivo:**

$$\begin{aligned} p(\mathbf{x} = \text{hello}) &= p(x_1 = h)p(x_2 = e | x_1 = h) \\ &\quad \times p(x_3 = l | x_1 = h, x_2 = e) \\ &\quad \times \cdots \times p(x_5 = o | x_1 = h, x_2 = e, x_3 = l, x_4 = l). \end{aligned}$$



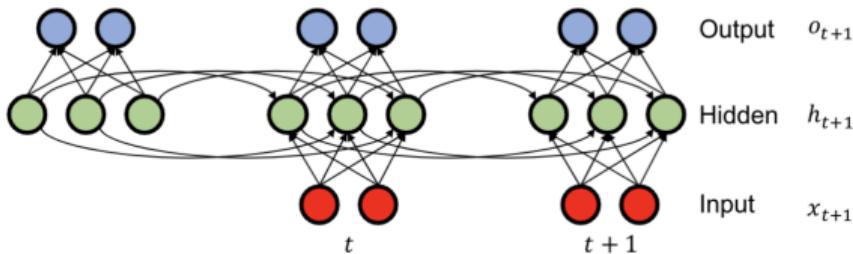
- Codificación one-hot:** $h \rightarrow (1, 0, 0, 0)$, $e \rightarrow (0, 1, 0, 0)$, $l \rightarrow (0, 0, 1, 0)$, $o \rightarrow (0, 0, 0, 1)$

- Ejemplo (figura):** $\mathbf{h}_1 = \tanh(W_{hh}\mathbf{h}_0 + W_{xh}\mathbf{x}_1)$

$$\mathbf{o}_1 = W_{hy}\mathbf{h}_1$$

$$p(x_2 = e | x_1 = h) = \text{softmax}(\mathbf{o}_1) = \frac{\exp(2.2)}{\exp(1.0) + \exp(2.2) + \exp(-3.0) + \exp(4.1)}$$

RNN: Redes Neuronales Recurrentes



- **Pros:**

- Se puede aplicar a secuencias de longitud arbitraria.
- Muy general: para cualquier función computable, existe una RNN finita que puede calcularla.

- **Contras:**

- Todavía requiere un orden
- Evaluación secuencial de la verosimilitud (muy lento para el entrenamiento)
- Generación secuencial (inevitable en un modelo autorregresivo)
- Puede ser difícil de entrenar (explosión o desvanecimiento del gradiente).

Ejemplo: RNN de Caracteres

- Entrenar una RNN de 3 capas con 512 nodos ocultos en todas las obras de Shakespeare.
- Luego muestrear del modelo:

KING LEAR: O, if you were a feeble sight, the courtesy of your law,
Your sight and several breath, will wear the gods
With his heads, and my hands are wonder'd at the deeds,
So drop upon your lordship's head, and your opinion
Shall be against your honour.

- **Nota:** la generación ocurre **carácter por carácter**. Necesita aprender palabras válidas, gramática, puntuación, etc.

Ejemplo: RNN de Caracteres (de Andrej Karpathy)

- Entrenar en Wikipedia. Luego muestrear del modelo:

Naturalism and decision for the majority of Arab countries' capitalide was grounded by the Irish language by [[John Clair]], [[An Imperial Japanese Revolt]], associated with Guangzham's sovereignty. His generals were the powerful ruler of the Portugal in the [[Protestant Immineners]], which could be said to be directly in Cantonese Communication, which followed a ceremony and set inspired prison, training. The emperor travelled back to [[Antioch, Perth, October 25—21]] to note, the Kingdom of Costa Rica, unsuccessful fashioned the [[Thrales]], [[Cynth's Dajoard]], known in western [[Scotland]], near Italy to the conquest of India with the conflict.

- **Nota:** sintaxis Markdown correcta. Apertura y cierre de corchetes [[·]]

Ejemplo: RNN de Caracteres

- Entrenar en Wikipedia. Luego muestrear del modelo:

– – cite journal — id=Cerling Nonforest Department—
format=Newlymeslated—none “ ”
"www.e-complete".

'''See also''': [[List of ethical consent processing]]

== See also ==

*[[lender dome of the ED]]

*[[Anti-autism]]

== External links==

* [http://www.biblegateway.nih.gov/entrepre/ Website of the World
Festival. The labour of India-county defeats at the Ripper of California
Road.]

Ejemplo: RNN de Caracteres

- Entrenar en un conjunto de datos de nombres de bebés. Luego muestrear del modelo:

Rudi Levette Berice Lussa Hany Mareanne Chrestina Carissy Marylen Hammine Janye Marlise Jacacie Hendred Romand Charienna Nenotto Ette Dorane Wallen Marly Darine Salina Elvyn Ersia Maralena Minoria El- lia Charmin Antley Nerille Chelon Walmor Evena Jeryly Stachon Charisa Allisa Anatha Cathanie Geetra Alexie Jerin Cassen Herbett Cossie Ve- len Dau- renge Robester Shermond Terisa Licia Roselen Ferine Jayn Lusine Charyanne Sales Sanny Resa Wallon Martine Merus Jelen Candica Wallin Tel Rachene Tarine Ozila Ketia Shanne Arnande Karella Roselina Alessia Chasty Deland Berther Geamar Jackein Mellisand Sagdy Nenc Lessie Rasemy Guen Gavi Milea Anneda Margoris Janin Rodelin Zeanna Elyne Janah Ferzina Susta Pey Castina

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basados en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

PixelRNN

PixelCNN

Transformers generativos

Aplicaciones

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basados en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

PixelRNN

PixelCNN

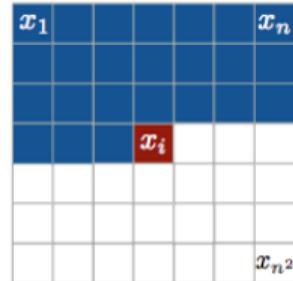
Transformers generativos

Aplicaciones

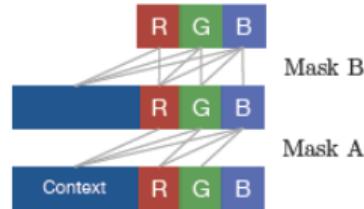
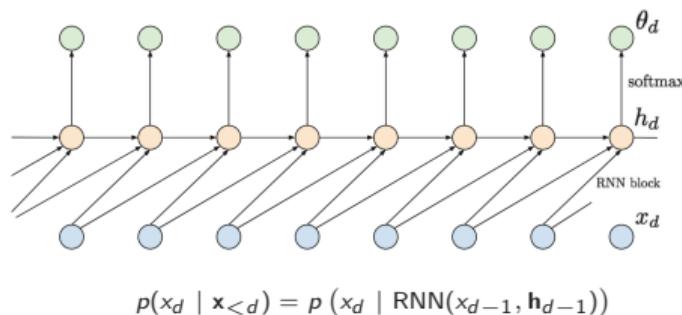
Pixel RNN

- Modelado pixel a pixel usando el orden de escaneo ráster.
- Cada condicional de pixel $p(x_t | \mathbf{x}_{1:t-1})$ necesita especificar 3 colores:

$$p(x_t | \mathbf{x}_{1:t-1}) = p(x_t^{\text{red}} | \mathbf{x}_{1:t-1}) \times p(x_t^{\text{green}} | \mathbf{x}_{1:t-1}, x_t^{\text{red}}) \\ \times p(x_t^{\text{blue}} | \mathbf{x}_{1:t-1}, x_t^{\text{red}}, x_t^{\text{green}}).$$

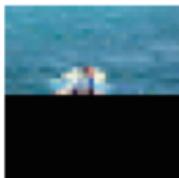


- Cada condicional es una V.A. categórica con 256 valores posibles.
- Condicionales modeladas usando variantes de RNN (LSTMs + masking tipo MADE)



Pixel RNN

occluded



completions



original



Resultados en ImageNet submuestreado. Muy lento: evaluación secuencial de la verosimilitud.

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basados en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

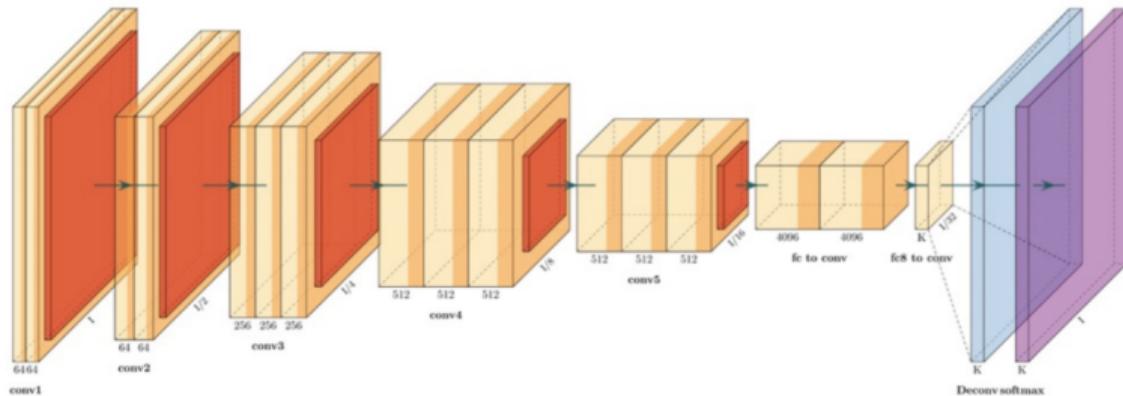
PixelRNN

PixelCNN

Transformers generativos

Aplicaciones

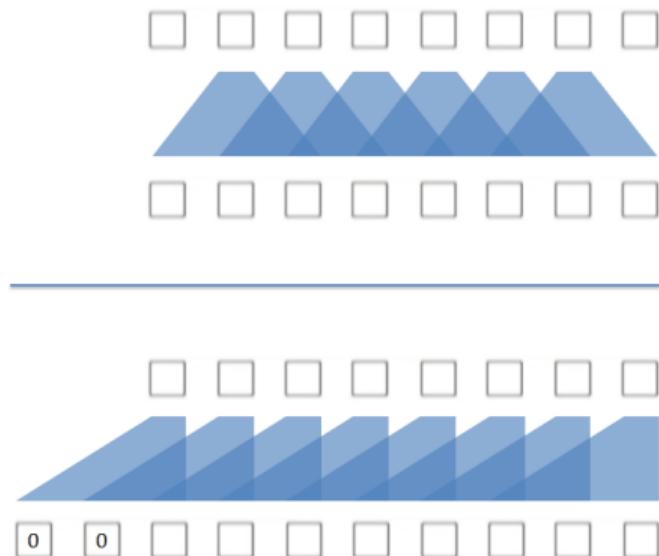
Arquitecturas Convolucionales



- La convolución es una operación invariante a traslaciones, bien adaptada para trabajar con imágenes. Son fáciles de paralelizar en hardware moderno.
- Las redes convolucionales alterna operaciones lineales (convolución) con no lineales (funciones de activación, poolings, etc.)

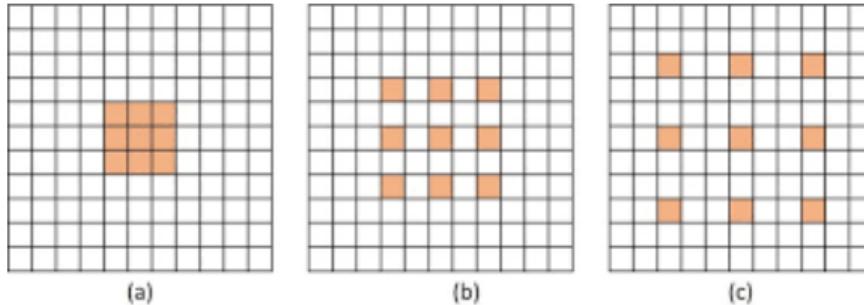
Convoluciones causales (ilustración 1D)

- La convolución en general (arriba) utiliza filtros de soporte simétrico $\Rightarrow x_{i+1}$ está dentro del campo receptivo involucrado en la predicción de \hat{x}_i (violación de la causalidad).



- La convolución causal (abajo) enmascara las partes del filtro que ven al futuro.

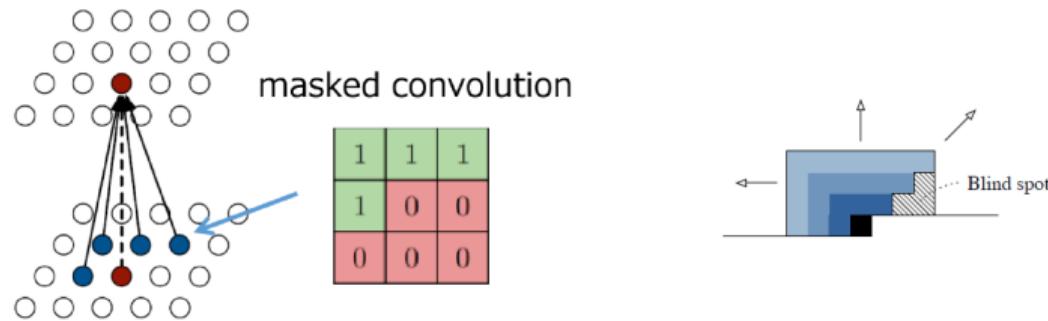
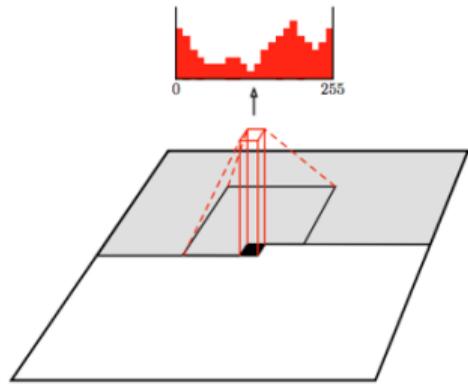
Convoluciones dilatadas



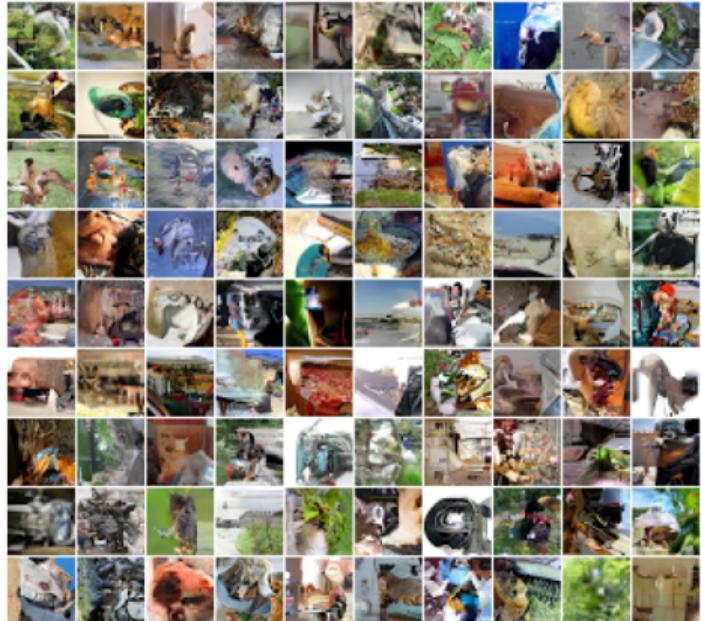
- Las convoluciones dilatadas aumentan el campo receptivo (aumentan el soporte, a misma cantidad de elementos del filtro).

Pixel CNN

- **Idea:** utilizar arquitectura convolucional para predecir el pixel siguiente en función del contexto (vecindario de píxeles).
- **Desafío:** debe ser **autorregresivo**
 - ⇒ Convoluciones enmascaradas (**causalidad**) conservan el orden de barrido.
 - ⇒ Enmascaramiento adicional para el orden de los colores.



Pixel CNN



- Muestras generadas con el modelo entrenado en ImageNet (32×32 pixels).
- Performance similar a PixelRNN pero mucho más rápido.

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basados en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

PixelRNN

PixelCNN

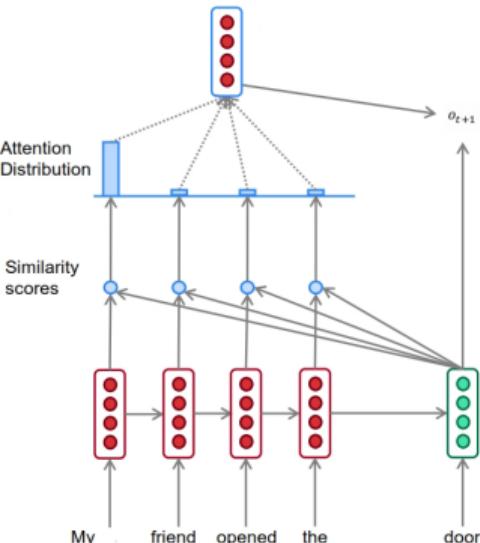
Transformers generativos

Aplicaciones

Modelos basados en atención

Mecanismo de atención para comparar un vector **Query** con un conjunto de vectores **Key**.

- ① Comparar el estado oculto actual (**Query**) con todos los estados ocultos pasados (**Keys**), por ejemplo calculando el producto escalar (similitud coseno)
- ② Construir la distribución de atención para ver qué partes del historial son relevantes, e.g., a través de un softmax
- ③ Construir un resumen del historial, por ejemplo, mediante una suma ponderada
- ④ Usar el resumen y el estado oculto actual para predecir el siguiente token/palabra



Transformers Generativos



Estado del arte (GPT, Bert): reemplazar RNN con Transformer

- Mecanismos de atención para enfocarse adaptivamente solo en el contexto relevante
- Evitar el cálculo recursivo. Usar solo módulos de *self-attention* para habilitar la paralelización
- Necesita **masked self-attention** para preservar la estructura autorregresiva.

① Repaso de la clase anterior

② Modelos autorregresivos básicos

Fully Visible Sigmoid Belief Network (FVSBN): modelo basado en regresión logística

NADE y RNADE: modelos basados en perceptrón multicapa

Autoencoders enmascarados como modelos autorregresivos

Redes Neuronales Recurrentes como modelos autorregresivos

③ Modelos autorregresivos recientes

PixelRNN

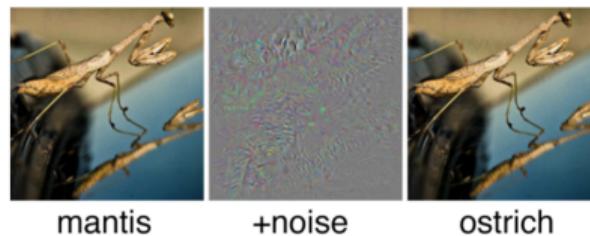
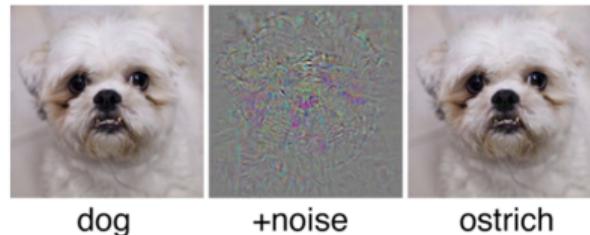
PixelCNN

Transformers generativos

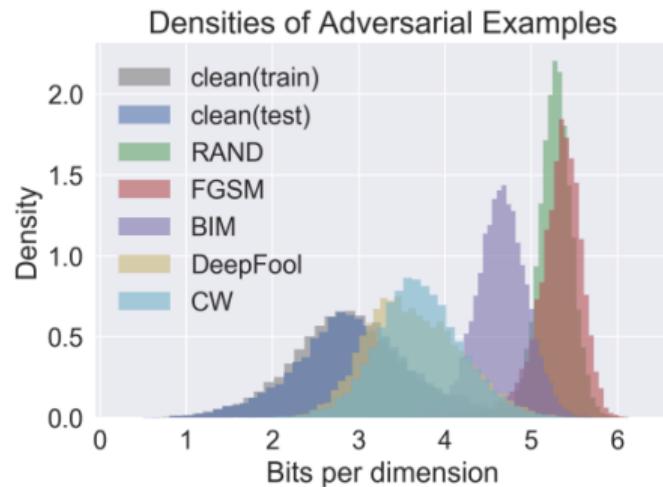
Aplicaciones

Aplicaciones en Ataques Adversarios y Detección de Anomalías

- Los métodos de aprendizaje son vulnerables a ejemplos adversarios



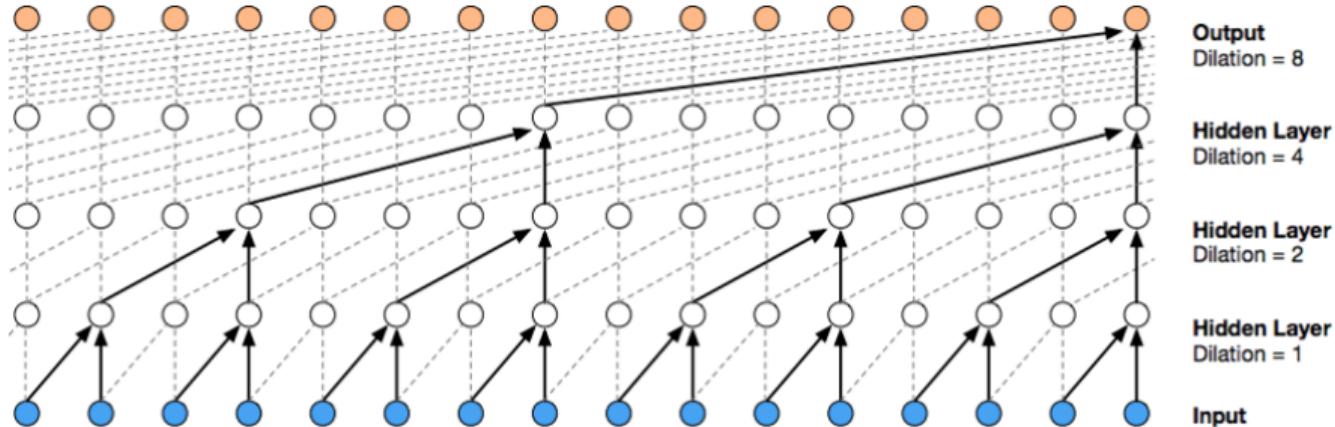
- Podemos detectarlos?



- Entrenar un modelo generativo $p(\mathbf{x})$ con entradas limpias (PixelCNN)
- Dada una nueva entrada $\bar{\mathbf{x}}$, evaluar $p(\bar{\mathbf{x}})$
- Los ejemplos adversarios son significativamente menos probables bajo $p(\mathbf{x})$

WaveNet

- Modelo muy efectivo para síntesis de voz (estado del arte):



- Las convoluciones dilatadas aumentan el campo receptivo: los filtros solo se aplican a la señal cada 2^d pixels.

Resumen de Modelos Autorregresivos

- Fácil de muestrear:
 - ① Muestrear $\bar{x}_0 \sim p(x_0)$
 - ② Muestrear $\bar{x}_1 \sim p(x_1 | x_0 = \bar{x}_0)$
 - ③ ...
- Fácil de calcular la probabilidad $p(\mathbf{x} = \bar{\mathbf{x}})$:
 - ① Calcular $p(x_0 = \bar{x}_0)$
 - ② Calcular $p(x_1 = \bar{x}_1 | x_0 = \bar{x}_0)$
 - ③ Multiplicar (sumar sus logaritmos)
 - ④ ...
 - ⑤ Se pueden calcular todos estos términos en paralelo para un entrenamiento rápido
- Fácil de extender a variables continuas. Por ejemplo, se pueden elegir condicionales Gaussianas $p(x_t | \mathbf{x}_{<t}) = \mathcal{N}(\mu_\theta(\mathbf{x}_{<t}), \Sigma_\theta(\mathbf{x}_{<t}))$ o mezcla de logísticas.
- No hay forma natural de obtener features para clustering o aprendizaje no supervisado.

Referencias

-  C. M. Bishop, *Pattern Recognition and Machine Learning*.
Springer, 2006.
-  Stanford, “CS236 Deep Generative Models.” <https://deepgenerativemodels.github.ioLecture>, 2023.
-  J. M. Tomczak, *Deep Generative Modeling*.
Springer Cham, 2024.