Inteligencia Artificial Generativa - Obligatorio 1

Matias Molinolo, Martina Diaz

Facultad de Ingeniería, Universidad ORT Uruguay

3 de octubre de 2023

https://github.com/matiasmolinolo/iag-2023

Índice

1. Estimación de distribuciones **3**
1.1. Dataset 1: Tennis 3
1.2. Dataset 2: Iris 3

2. Modelos generativos **3**
2.1. NADE: Neural Autoregressive Distribution Estimator 3
2.2. VAE: Variational Autoencoder 5

1. Estimación de distribuciones

1.1. Dataset 1: Tennis

El objetivo fue estimar distribuciones de probabilidad a partir de un dataset 'Tennis' con datos discretos. Inicialmente se calculó la probabilidad conjunta $p(x, y)$ y las probabilidades condicionales $p(y|x)$ y $p(x|y)$, siendo x la variable discreta 'Outlook' e y la variable booleana 'Tennis'.

1.2. Dataset 2: Iris

marti

2. Modelos generativos

2.1. NADE: Neural Autoregressive Distribution Estimator

Intentamos reproducir en Pytorch la arquitectura del NADE propuesta por [1], con ciertas modificaciones que resultaron de la prueba y error de la arquitectura.

Uria et al. propone una arquitectura autorregresiva que parte de la base que cualquier distribución D-dimensional p puede expresarse como un producto de distribuciones unidimensionales [1]:

$$p(\boldsymbol{x}) = \prod_{d=1}^{D} p(x_{o_d} \mid \boldsymbol{x}_{o_{<d}})$$

Para ello, nos basamos en el siguiente repositorio de Github: https://github.com/simonjisu/NADE-pytorch/blob/master/model.py, pero con ciertas modificaciones para la optimización de los cálculos.

Nuestra arquitectura usa 192 unidades ocultas, en vez de las 500 propuestas en el paper original. Esto se debe a que tras varios ensayos, encontramos que este número daba una buena performance y un tiempo de ejecución significativamente menor al de la arquitectura original del paper.

El NADE fue entrenado con datos binarizados del dataset MNIST, donde se filtraron todos aquellos elementos que no pertenecían a la clase que quería generarse.

Podemos ver el código de la función `forward` del NADE a continuación:

```
a = self.c.expand(x.shape[0], -1)

p_x_hat = []

for d in range(self.input_dim):
    h_d = torch.sigmoid(a)
    p = torch.sigmoid(h_d @ self.V[d:d+1, :].t() + self.b[d:d+1])
    p_x_hat.append(p)
    a = x[:, d:d+1] @ self.W[:, d:d+1].t() + a

probs = torch.cat(p_x_hat, 1)
return probs
```

Luego de entrenarlo por 50 epochs, donde alcanzamos una loss de 0,29, hicimos un muestreo de la distribución estimada por el NADE con el siguiente código:

```
    with torch.no_grad():
    preds = torch.zeros(1, input_dim).to(device)
    for i in tqdm(range(input_dim)):
        p = model.forward(preds)
        preds[0][i] = torch.bernoulli(p[0][i])

    return torch.reshape(preds.cpu(), (28, 28))
```

Donde obtuvimos el siguiente resultado:

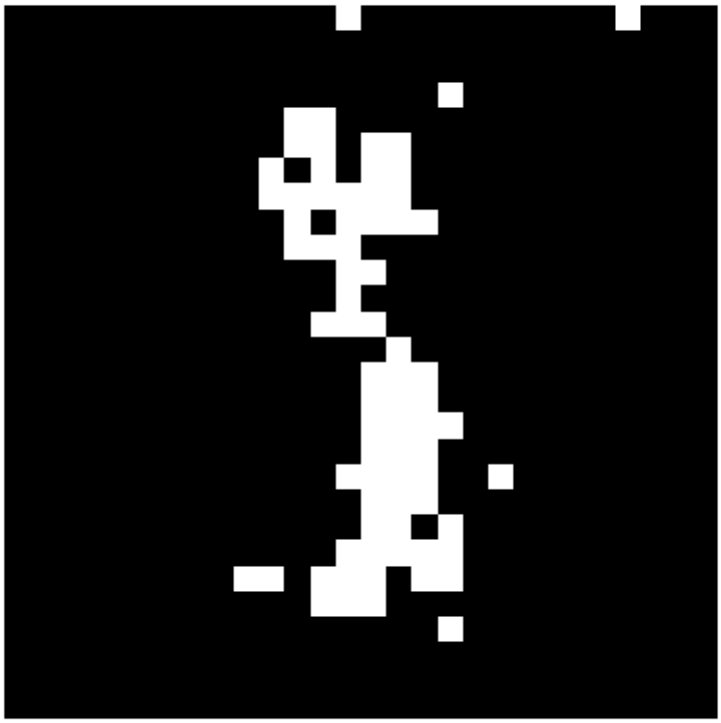


Figura 1: Distribución estimada por el NADE

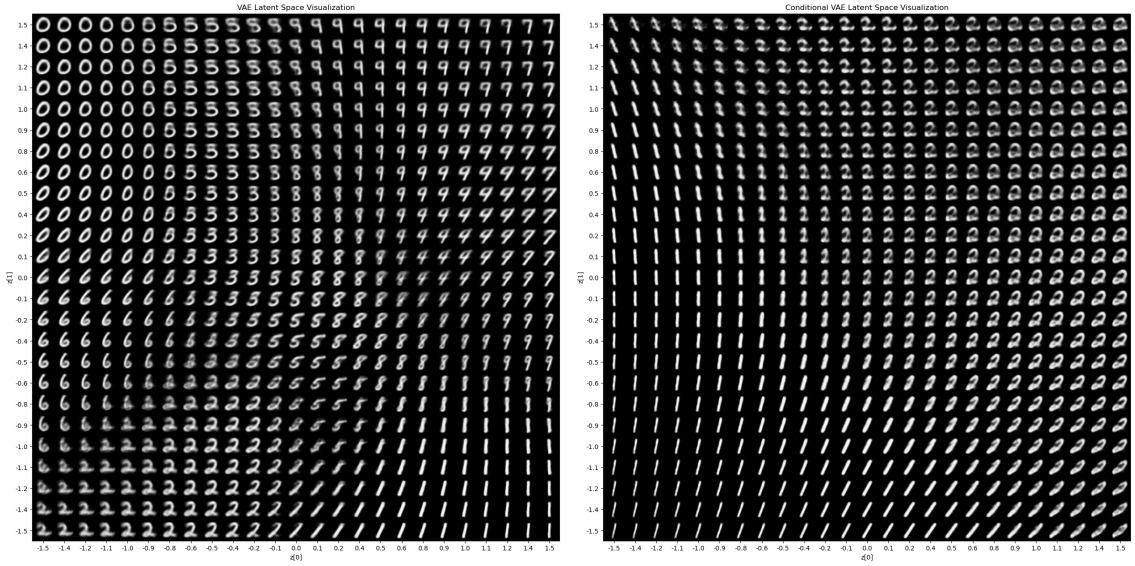
Podemos ver que no es una representación totalmente acertada, pero se puede apreciar claramente que se asemeja a la clase original de los 1. Estimamos que esto puede deberse a falta de entrenamiento u otros factores.

2.2. VAE: Variational Autoencoder

A partir del tutorial de Doersch [2], entrenamos un VAE y un VAE condicional, es decir, que genera una nueva imagen dada una clase específica.

Nuevamente, entrenamos esta red con el dataset del MNIST, pero esta vez sin filtrar ni binarizar.

Es interesante visualizar el espacio latente del VAE y el VAE condicional, que podemos observar en las siguientes figuras:



(a) Espacio latente del VAE

(b) Espacio latente del VAE condicional

Podemos apreciar como en el espacio latente del VAE se generan nuevas imágenes que se asemejan a todas las clases del dataset, mientras que en el espacio latente del VAE condicional se generan nuevas imágenes acotadas a la clase que indicamos como condicional.

Referencias

[1] B. Uria, M.-A. Côté, K. Gregor, I. Murray, and H. Larochelle, "Neural autoregressive distribution estimation," 2016.

[2] C. Doersch, "Tutorial on variational autoencoders," 2021.