Informe Final: Bayesian Flow Networks

Integrantes: Arturo Lazcano y Javier Santidrián **Profesor:** Felipe Tobar
Auxiliares: Cristóbal Alcázar y Camilo Carvajal Reyes **Ayudante:** Joaquín Barceló **Fecha:** 12/12/2023

Introducción

Este proyecto se basa en explicar y aplicar Bayesian Flow Networks (BFN) [1] como algoritmo de generación de datos. En el paper se analiza esto para data continua y discreta, pero el enfoque de este proyecto será en data discreta, en particular, generación de dígitos (MNIST modificado).
Este algoritmo se basa en inferencia Bayesiana y datos ruidosos, es decir, datos que se les aplica una transformación con tal de que el modelo pueda aprender a quitarle ese ruido convergiendo a una distribución, lo cual en este caso es un dígito.
Existen algunas similitudes entre BFNs y modelos de difusión, pero la principal diferencia es que estos últimos operan sobre la misma data con ruido, mientras que BFNs operan sobre los parámetros de la distribución de los datos.

Teoría

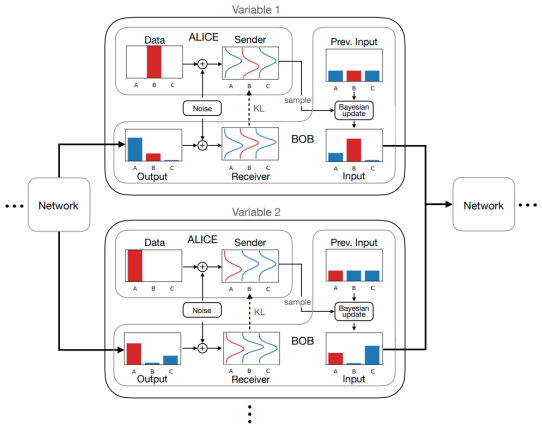

Figura 1: Analogía BFNs.

En la figura anterior se explica BFN con la analogía de Alice y Bob. En ella, Bob tiene una *input distribution* que al principio es un prior (uniforme en caso discreto), el cual le pasa sus parámetros a la red para obtener la *output distribution*. Por otro lado, Alice crea una *sender distribution* añadiendo ruido a la data real, mientras que Bob crea una *receiver distribution* usando la *output distribution* y la misma distribución de ruido que usó Alice. Luego, Alice samplea de la *sender distribution* y se lo envía a Bob con un costo de divergencia KL entre la *sender* y *receiver* respectivamente. Así, Bob usa ese sampleo para actualizar su *input distribution* usando la regla de Bayes. Luego, Bob nuevamente alimenta los parámetros de la *input distribution* a la red que retorna los parámetros de la *output distribution*. Este proceso se repite n veces.

Algorithm 9 Sample Generation for Discrete Data

Require: $\beta(1) \in \mathbb{R}^+$, number of steps $n \in \mathbb{N}$, number of classes $K \in \mathbb{N}$
$\boldsymbol{\theta} \leftarrow \left(\frac{1}{K}\right)$
for $i = 1$ to n **do**
  $t \leftarrow \frac{i-1}{n}$
  $\mathbf{k} \sim \text{DISCRETE\_OUTPUT\_DISTRIBUTION}(\boldsymbol{\theta}, t)$
  $\alpha \leftarrow \beta(1)\left(\frac{2i-1}{n^2}\right)$
  $\mathbf{y} \sim \mathcal{N}\left(\alpha\left(K\mathbf{e}_{\mathbf{k}} - \mathbf{1}\right), \alpha K \boldsymbol{I}\right)$
  $\boldsymbol{\theta}' \leftarrow e^{\mathbf{y}}\boldsymbol{\theta}$
  $\boldsymbol{\theta} \leftarrow \frac{\boldsymbol{\theta}'}{\sum_k \boldsymbol{\theta}'_k}$
end for
$\mathbf{k} \sim \text{DISCRETE\_OUTPUT\_DISTRIBUTION}(\boldsymbol{\theta}, 1)$
Return k

Figura 2: Algoritmo de generación de datos discretos.

En la figura anterior se muestra el pseudocódigo del algoritmo para generación de muestras de datos discretos. Este comienza inicializando un vector θ con probabilidades uniformes para representar la falta de conocimiento previo sobre las K clases posibles. A lo largo de n pasos iterativos, el algoritmo avanza a través del tiempo normalizado t, generando muestras discretas k y ajustando la precisión α mediante una función programada β que cambia en cada paso. Las muestras se extraen de una distribución normal influenciada por la clase actual k, incorporando ruido controlado al proceso. Los parámetros de θ se actualizan exponencialmente en base a las muestras y luego se normalizan para mantener una distribución de probabilidad válida. Tras este bucle, se genera la muestra final k de la distribución de salida discreta basada en los parámetros finales θ y el tiempo 1 (fin del proceso).

Metodología

Se utilizará el algoritmo anterior con el dataset discreto MNIST (70 mil imágenes de dígitos del 0 al 9 escritos a mano de 28x28, donde 60 mil de ellos son para entrenamiento y 10 mil para prueba) dinámicamente binarizado, es decir, el dataset original pero donde los valores de los píxeles se convierten de manera estocástica en binarios (0 o 1) cada vez que se carga un lote de datos durante el entrenamiento (lo cual añade variabilidad o ruido sobre la data).
Por otro lado, se hará el mismo experimento junto con una modificación del dataset MNIST. Esta modificación consiste en usar 2 dígitos en lugar de 1, es decir, cada dato (imagen) consistirá en dos dígitos donde, por motivos del algoritmo de BFN, también estarán binarizados por lo que cada píxel posee valores 0 o 1.

Finalmente, para implementar la BFN se utilizará una arquitectura de red U-Net basada en modelos de difusión con la cual se generarán nuevos datos.

Datasets

Para la implementación original de la BFN se utiliza data discreta como es el MNIST. Esto para comprobar el buen funcionamiento del algoritmo junto con obtener una noción tanto de la arquitectura como la complejidad y el tiempo de entrenamiento de la red.
Para el uso de MNIST, se debe cargar este dataset en formato de imágenes de 28x28 donde es binarizado mediante el uso de un vector de números aleatorios entre 0 y 1 (vector con probabilidades en cada componente) y asociando uno de estos puntos a cada probabilidad y ver si este resultará en un píxel negro o blanco, es decir, si un punto arbitrario x_i de la imagen posee valor mayor que p_i, este se verá reflejado como un píxel blanco (1). En el caso que $x_i \leq p_i$, será un píxel negro (0).
Un dato de este dataset se puede ver en la siguiente figura:

Figura 3: Dato aleatorio de MNIST.

Por otro lado, el aporte a este trabajo realizado por Graves et al.[1] es usar un nuevo dataset. Este consiste en dígitos al igual que el MNIST usual, sin embargo, en cada imagen aparecen 2 dígitos en lugar de 1.
La forma en que estas imágenes son generadas está en el github oficial de este dataset [2]. Notar que cada imagen será de 64x64 y no de 28x28 como el MNIST usual. Esto es debido a que cada dato se genera usando 2 imágenes del MNIST y concatenándolas de cierta forma, por lo que si nuestras imágenes son de 28x28, necesitamos que estos nuevos datos sean de al menos 64x64, lo que ya entrega un indicio de que la red puede ser más ineficiente a la hora de entrenar pues tendrá mucho trabajo más que aprender que en la imagen más pequeña.
Nuevamente, un dato de este dataset se puede ver en la siguiente figura:

Figura 4: Dato aleatorio de MNIST modificado.

Red Neuronal

Como se dijo en secciones anteriores, la red neuronal a entrenar y usar en la BFN será una UNet, la cual es una red con buenos resultados en datos que involucren imágenes.
A modo de poder ejecutar esta red, se usa una versión simplificada que se puede describir en la siguiente imagen:

```
UNet(
  (e11): Conv2d(2, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dpe11): Dropout(p=0.5, inplace=False)
  (e12): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dpe12): Dropout(p=0.5, inplace=False)
  (e13): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (e21): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dpe21): Dropout(p=0.5, inplace=False)
  (e22): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dpe22): Dropout(p=0.5, inplace=False)
  (e23): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (e31): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dpe31): Dropout(p=0.5, inplace=False)
  (e32): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dpe32): Dropout(p=0.5, inplace=False)
  (e33): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (upconv1): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2))
  (d11): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dpd11): Dropout(p=0.5, inplace=False)
  (d12): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dpd12): Dropout(p=0.5, inplace=False)
  (d13): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (upconv2): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
  (d21): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dpd21): Dropout(p=0.5, inplace=False)
  (d22): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (dpd22): Dropout(p=0.5, inplace=False)
  (d23): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
  (outconv): Conv2d(64, 1, kernel_size=(1, 1), stride=(1, 1))
)
```

Figura 5: UNet simplificada.

Esta arquitectura más simple que la versión original permite tener menos parámetros a entrenar por lo que el entrenamiento será un poco más rápido.
La implementación tanto de esta red como del algoritmo de Bayesian Flow Network viene inspirada de [3].

Resultados

El entrenamiento de ambos modelos (uno para cada dataset) fue exitoso. La evolución de las función de pérdida que tuvo cada modelo se puede ver en la siguiente imagen.

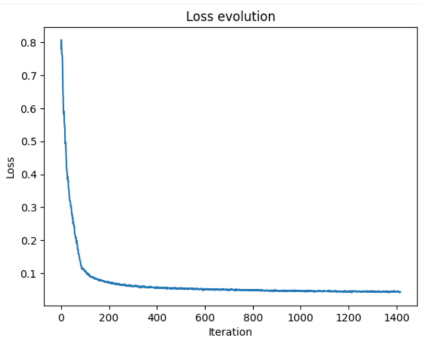

Figura 6: Trayectoria de loss para MNIST.

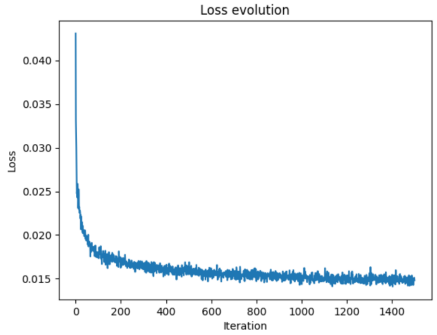

Figura 7: Trayectoria de loss para MNIST modificado.

Ambos modelos tuvieron una implementación exitosa, sin embargo, debido a la capacidad computacional y restricciones de tiempo, no se pudo perfeccionar para dar unos resultados comparables a los datos originales. Esto se puede ver en las siguientes figuras donde se puede apreciar un sampleo post-entrenamiento de 9 datos tanto para el dataset de MNIST como su modificación.

Figura 8: Sampleo de datos para MNIST.

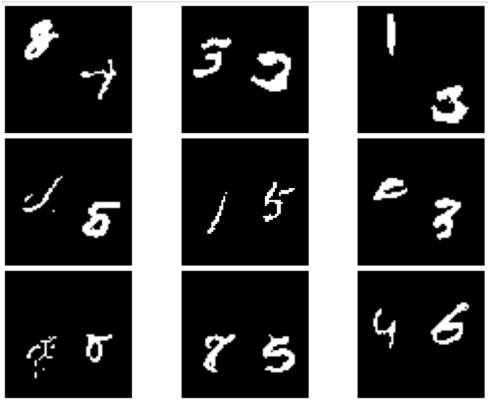

Figura 9: Sampleo de datos para MNIST modificado.

Por último, se presenta en las figuras 10 y 11 la trayectoria de un solo dato y cómo este parte siendo solo ruido y bajo modificaciones sutiles a la distribución original se empiezan a formar algún dígito tanto para la BFN con MNIST como la modificación de este dataset.

Trabajo a Futuro

Como se puede observar en la sección de resultados, aún falta entrenamiento de la red para que pueda generar mejores samples. Esto toma más tiempo y poder computacional por lo que la plataforma de google colaboratory no sirve para ejecutar todo pues se queda sin recursos.
Por otro lado, se espera una comparación con modelos de difusión debida a la parecida naturaleza que ambos modelos tienen al tratar imágenes con ruido como forma de reconstruir, y generar, nuevos datos.
Por otro lado, se espera el uso de otros dataset para evaluar el funcionamiento de las BFN.

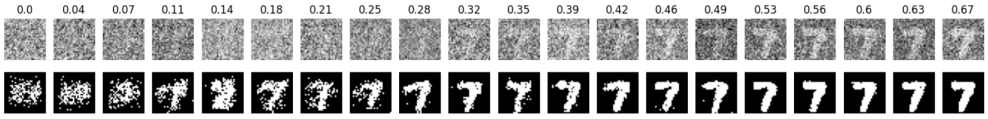

Figura 10: Procesamiento de imagen en la BFN para MNIST.

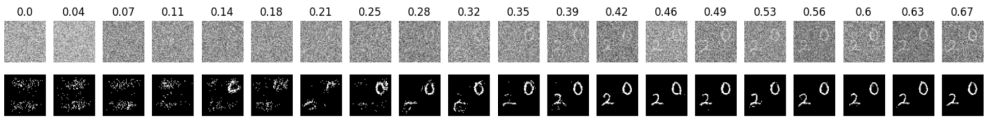

Figura 11: Procesamiento de imagen en la BFN para MNIST modificado.

Referencias

[1] A. Graves, R. K. Srivastava, T. Atkinson, and F. Gomez, "Bayesian flow networks," *arXiv preprint arXiv:2308.07037*, 2023.

[2] S.-H. Sun, "Multi-digit mnist for few-shot learning." https://github.com/shaohua0116/MultiDigitMNIST, 2019.

[3] A. Hibble, D. Ghilardi, and A. Turner, "Bayesian flow networks." https://github.com/Algomancer/Bayesian-Flow-Networks, 2023.