TP5 - Deep Learning

Integrantes

- Rodrigo Navarro
- Florencia Monti
- Guido Princ

01
INTRODUCCIÓN

02
AUTOENCODERS

03
DEMOSTRACIÓN

04
RESULTADOS Y CONCLUSIONES

Introducción

01

Se nos pidió implementar un autoencoder básico y otro denoising para imágenes binarias y para otro conjunto de datos a elección.

Rodri

02

Autoencoders

Utilizamos un Perceptron Multicapa, donde la capa intermedia es de 2

Utilizamos Kohonen para la generación de letras y mapas

Tope de aprendizaje: ~20

Ej 2: mapas de Sokoban como conjunto de datos

Guido

Creación del MLP

Como base tomamos el MLP del TP3 y le agregamos una tasa de aprendizaje variable que no habíamos hecho. Nos pareció poco intuitivo tener que estar creando la longitud de la red a mano con las neuronas por lo que implementamos una clase llamada Layar Creator, la cual después de cada iteración ajusta y crea una red nueva aumentando o decreciendo la longitud de la red y a su vez también teníamos una cota que variaba al igual que la longitud la cual determinaba la mínima cantidad de neuronas por capa. Además por cada iteración los pesos se generan aleatoriamente.

Flor

Autoencoder Generativo

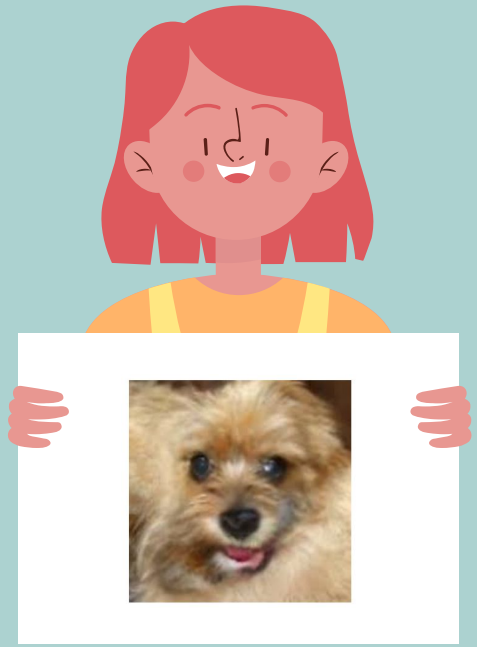
Mapa base

Conjunto de mapas cuadrados válidos creados específicamente para el aprendizaje.

-> Kohonen -> Decoder ->

Mapa nuevo

Mapa que queremos que sea jugable



Rodri

03

Demostración

Guido

04

Resultados

Respecto a la cantidad de letras aprendidas pudimos notar que se aprende como máximo a tiempo observable 21 patrones.

Luego con respecto al denoising depende mucho de la red pero encontramos redes que pueden hacer un denoising de 1 bit sin ningún problema con 5 patrones aprendidos mientras que estos sean lo suficientemente ortogonales del resto.

Los mapas que obtuvimos, pudimos notar que no ajustan a las reglas de los mapas del sokoban, por ej más cajas que win points.
Se nos ocurrió que hubiera estado mejor tener un “árbitro” que pueda entrenar al autoencoder diciéndole que mapas son viables y cuáles no.

Flor

Resultados

Aprendizaje de mapas

Capa del medio 80

```
********** Layers **********

128 122 118 86 93 109 99 112 83 78 124 80 124 78 83 112 99 109 93 86 118 122 128

=> Error = 0.39517346791087005
```

Capa del medio 75

```
********** Layers **********

128 83 119 96 76 112 99 77 85 127 99 75 99 127 85 77 99 112 76 96 119 83 128

 => Error = 1.294991841080445
```

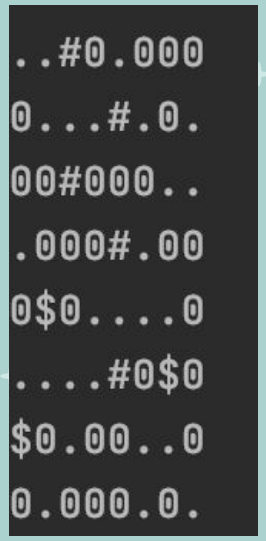
Rodri

```
MultiLayer Perceptron

********** Initialized font **********


********** Layers **********

35 31 25 33 21 24 2 24 21 33 25 31 35
 => Error = 0.32045032904504467
 => Error = 0.3206016484695294
 => Error = 0.32073573297984403
 => Error = 0.3208677738548615
 => Error = 0.32099785227654876
 => Error = 0.32113995273919727
 => Error = 0.3212663072960217
 => Error = 0.3213906812149541
 => Error = 0.321526820872294
 => Error = 0.32164765266247164
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0
0 1 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0
1 0 0 0 0 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 1 1 0 0 0 0 1 0 0 0 0
0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0
0 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 0 1 0 0 0 0 1 0 1 1 0 0 0 0 0 0 1 0 0

 => Error Average = 0.3210624751410767


********** Noise err **********

Err 0.09700357474915128

0 1 0 0 1 0 1 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 0 0 1 0 1 0 0 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
{
  "threshold": "0.1",
  "accuracy": "0.001",
  "mlp_lrate_even": "0.001",
  "mlp_iter_even": "500",
  "noise_percentage": "1",
  "ej": "1-noise",
  "font": "1",
  "letters": "5",
  "activation_method": "0",
  "layer_size": "5",
  "min_neurons": "TODO",
  "middleLayer": "2",
  "kohonen_k": "3",
  "kohonen_lr": "0.01",
  "kohonen_delta": "2",
  "error": "1",
  "lr_a": "0.001",
  "lr_b": "0.1",
  "iterations": "10",
  "mapSize": "8"
}
```

Guido

¡Mapas!



********** New output generated via kohonen weights **********

1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 1 0 1 0 0 0
0 0 0 0 1 1 0 0 0 0 0 0 0 1 1 0 0 1 0 1 0 1 0 1 0 0 0 1 0 1 0 1 0 1 0 0 1 0 0 1 1 0 0 1 1 0 0 1 0 0 0 0 0
1 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 1 0

05

Conclusiones

Flor

Patrones aprendidos

Aunque solo pudimos obtener a tiempos cortos de corridas un tope de 21 letras aprendidas, si se deja el algoritmo corriendo más tiempo capaz él mismo encontraba una combinación de capas, neuronas y pesos iniciales que hiciesen que el MLP pueda aprender más patrones.

Flor

Denoising

Con nuestra experiencia nos dimos cuenta lo importante que es entrenar bien el autoencoder y optimizarlo, aunque el nuestro sea muy básico se pudo defender ante la situación y lograrlo con un margen de 1 bit de error. Por lo tanto puede ser una gran herramienta de uso si se la optimiza mejor y hace un seguimiento de pesos y algoritmia.

Rodri

Ortogonalidad

Suponemos que la razón por lo que aprende algunas letras y otras no, se debe a la ortogonalidad de la misma. Ya que hay letras que las aprende con un error muy bajo mientras que otras se las confunde.

Guido

Pesos sinápticos

Algo que nos dimos cuenta que es un factor muy importante es la inicialización de pesos ya que esto puede cambiar totalmente el flujo de actualización de los mismos a partir del gradiente descendiente y por lo tanto no llegará nunca a converger en una solución con error aceptable.

Flor

Autoencoder Generativo

No pudimos lograr una compresión como hacíamos con las letras pero pudimos comprimirlos hasta 80 con un buen error de cota < 1. Pero al usarlo como un decoder generativo aun así no pudimos obtener mapas que pudieran ser jugables por el sokoban. Si hubiéramos implementado mecanismos más complejos de optimización y correr a tiempos largos de epoch que nuestras compus no podían, capas podríamos generar un autoencoder mejor que pueda realmente generar mapas aleatorios que se ajusten a las reglas del sokoban y que jamás se nos hubiesen ocurrido.

Rodri

Mapas Sin solución

Al definir los diferentes elementos de los mapas de sokoban con pocos bits (2 sin contar la posición del jugador) y no tener una capa extra que valide que los mapas generados sean válidos para jugar, obtuvimos mapas inválidos para jugar por lo que se requiere otra capa que provea esta información.

Guido

Formas de optimización

Al momento de aplicarle al autoencoder una taza de aprendizaje variable pudimos notar lo siguiente (aprendizaje alpha = 0.001).

TAZA VARIABLE

Usamos la fórmula vista en clase y con los valores A = 0.001 y B = 0.1

TAZA FIJA

Aca el valor se mantiene estático y no iba cambiando

Flor

Formas de optimización

Al momento de aplicarle al autoencoder generativo una taza de aprendizaje variable pudimos notar lo siguiente (aprendizaje alpha = 0.001).

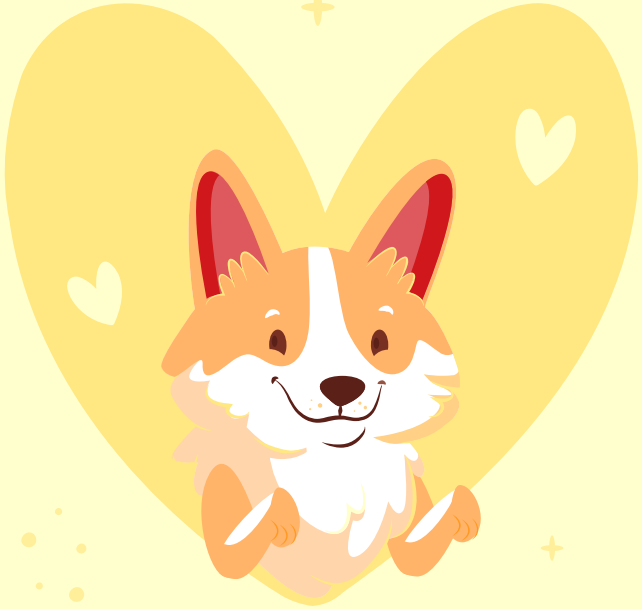
TAZA VARIABLE

Usamos la fórmula vista en clase y con los valores A = 0.0001 y B = 0.1

TAZA FIJA

Con otros valores de aprendizaje nunca logramos hacer que converja en solución

Rodri



¡Gracias!

¡A mimir!