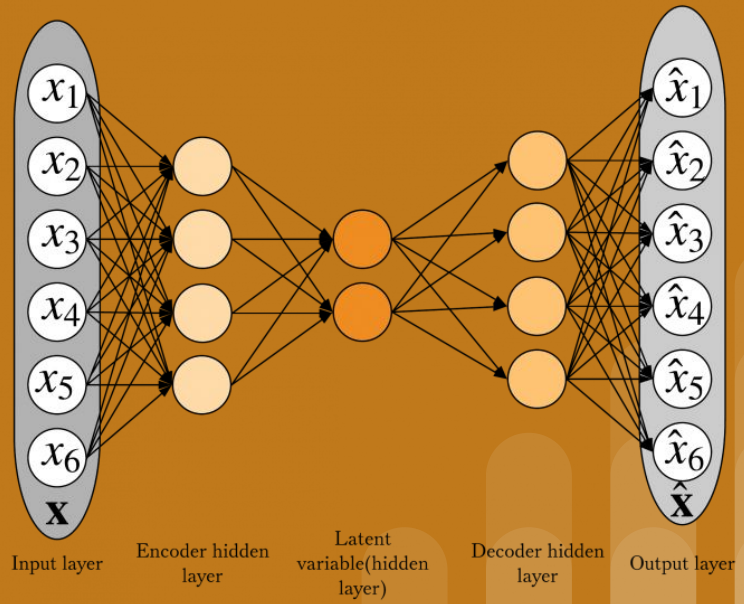
Trabajo Práctico N° 5: Deep Learning

Grupo 10

Autoencoder

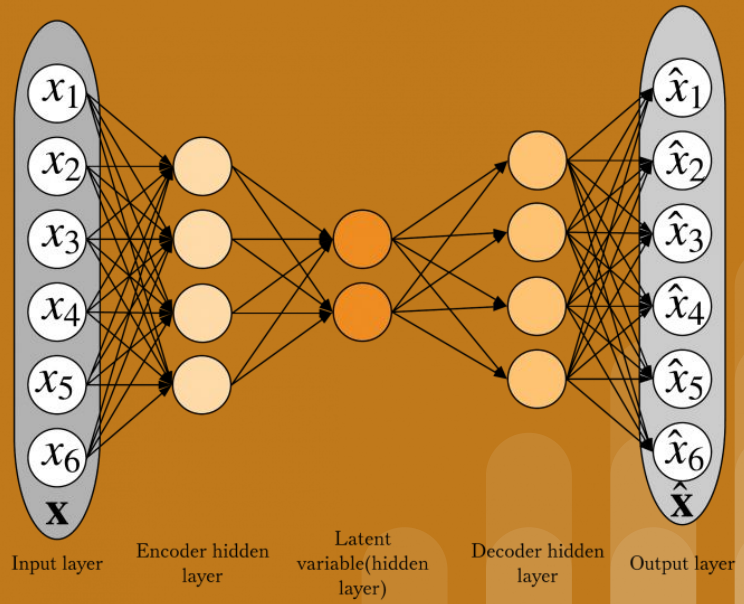
Autoencoder - Implementación

- Un único MLP simétrico
- Variación de capas y funciones de activación
- Espacio latente bidimensional
- Variación de capas intermedias (número y neuronas)
- Variación de learning rate
- Variación de epochs



Autoencoder - Optimización

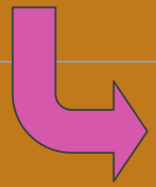
- Learning rate adaptativo
- Gradiente descendiente
- Momentum
- Gradience desdenciende estocástico
- Funciones de activación rectificadoras
- Combinaciones de métodos



Autoencoder - Optimización

Distribución capas internas	Error
{64,16,2,16,64}	0.355
{30,20,2,20,30}	0.267
{16,2,16}	1.373
{64,32,16,2,16,32,64}	0.003

Distribución capas internas	Error
{32, 2, 32}	0.802
{32, 8, 2, 8, 32}	0.163
{30, 24, 2, 24, 30}	0.448
{64, 2, 64}	0.511

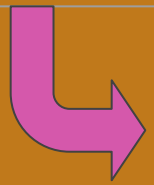


Mayor tiempo, mucho menos error

Autoencoder - Optimización

# Experimento	Error (momentum 0.7)
1	0.027
2	0.065
3	0.100
4	0.066

# Experimento	Error (sin momentum)
1	0.097
2	0.065
3	0.129
4	0.128



Mejor utilizar momentum

Autoencoder - Optimización

# Experimento	Error (eta adaptativo)
1	0.034
2	0.129
3	0.097
4	0.134

Disminuye (pero no tanto como momentum)

intentamos combinar

# Experimento	Error (eta adaptativo + momentum)
1	0.034
2	0.035
3	0.066
4	0.054

Autoencoder - Optimización

¿Y si añadimos gradiente descendiente estocástico?

# Experimento (Batch n = 5)	Error (eta adaptativo)
1	1.405
2	2.759
3	2.978
4	2.863

# Experimento (Batch n = 10)	Error (eta adaptativo)
1	0.355
2	0.532
3	0.692
4	0.423

No siempre disminuye

Autoencoder - Optimización

¿Y con distintas funciones de activación?

#	TanH (beta = 2)
1	0.034
2	0.035
3	0.066
4	0.054

*Buena opción, consistente

#	ELU (beta = 0.1)
1	0.023
2	17.26
3	0.026
4	0.028

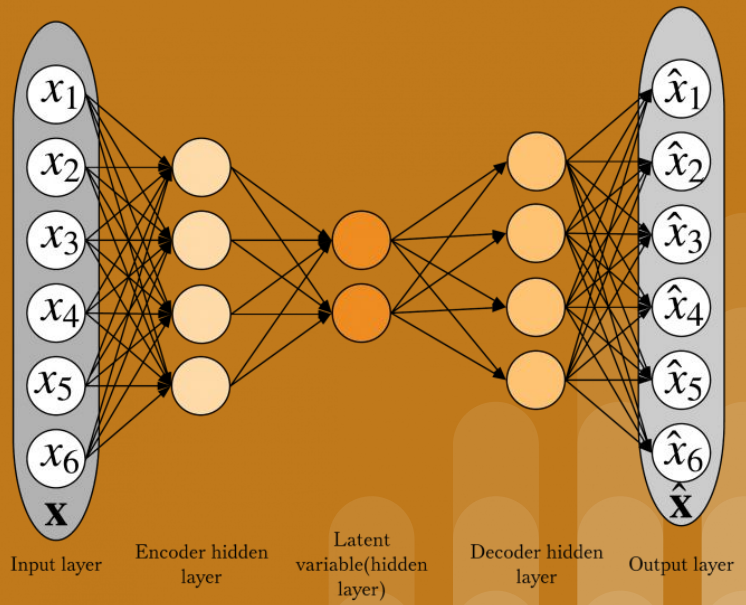
*Puede dar una activación muy grande y generar mucho error (inconsistente)

#	Leaky ReLU (beta = 0.01)
1	7.117
2	0.017
3	0.037
4	4.065

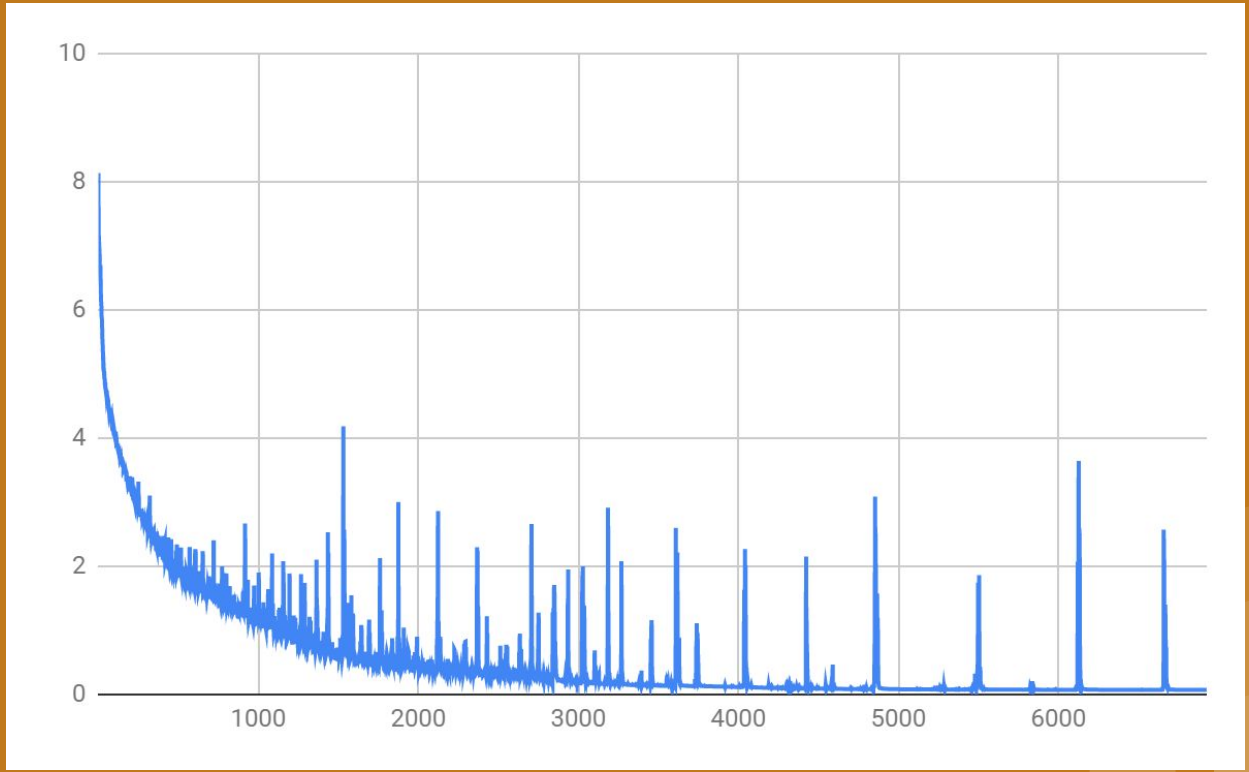
*En ciertos casos necesitaría más epochs para alcanzar el mi resultado que TanH

Autoencoder - Espacio latente

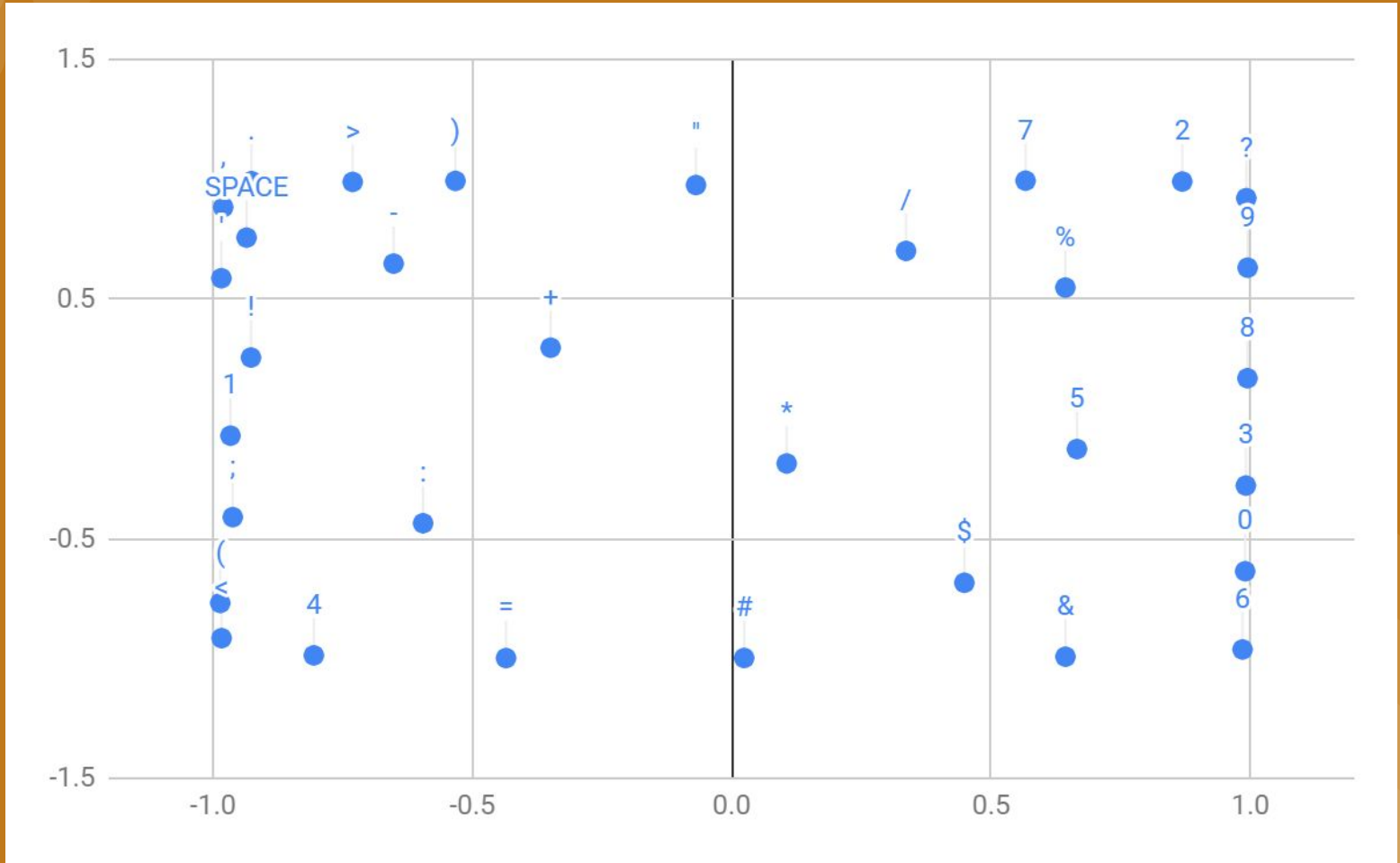
- Función de activación TanH
- 10000 Epochs (se puede disminuir el error con más)
- Distribución de capas internas {64, 32, 16, 2, 16, 32, 64}
- Momentum término 0.7
- Learning rate adaptativo comenzando en 0.001



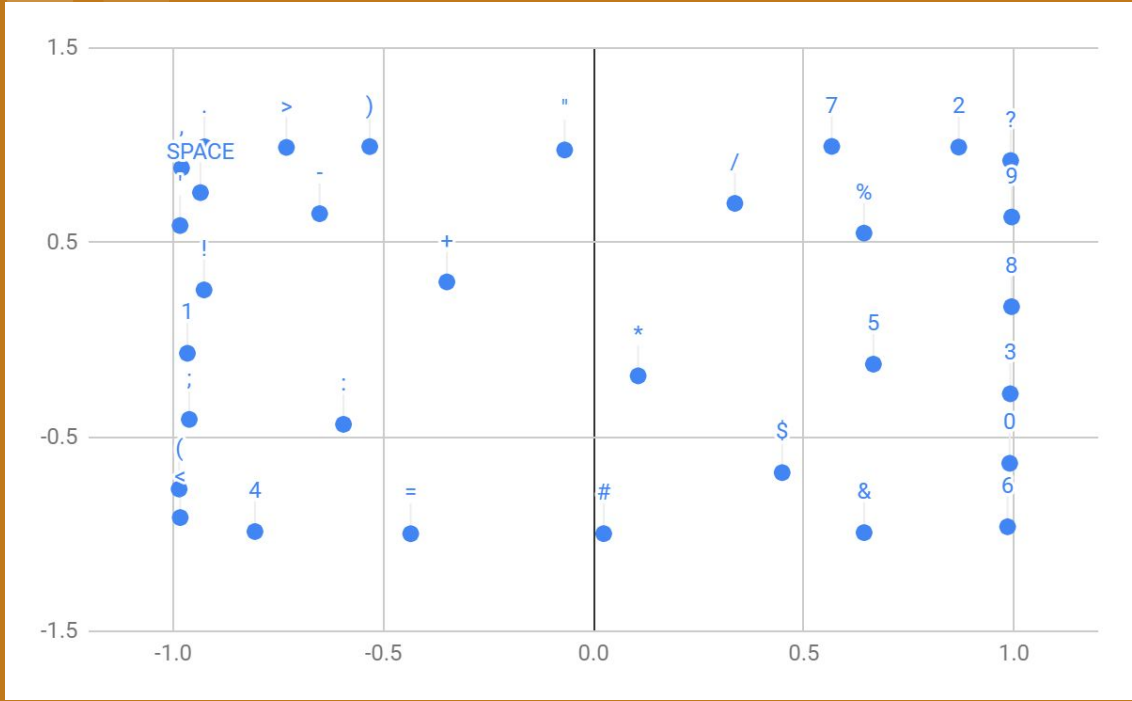
Autoencoder - Disminución del error (err mín 0.07)



Autoencoder - Espacio latente



Autoencoder - Generando nueva letra



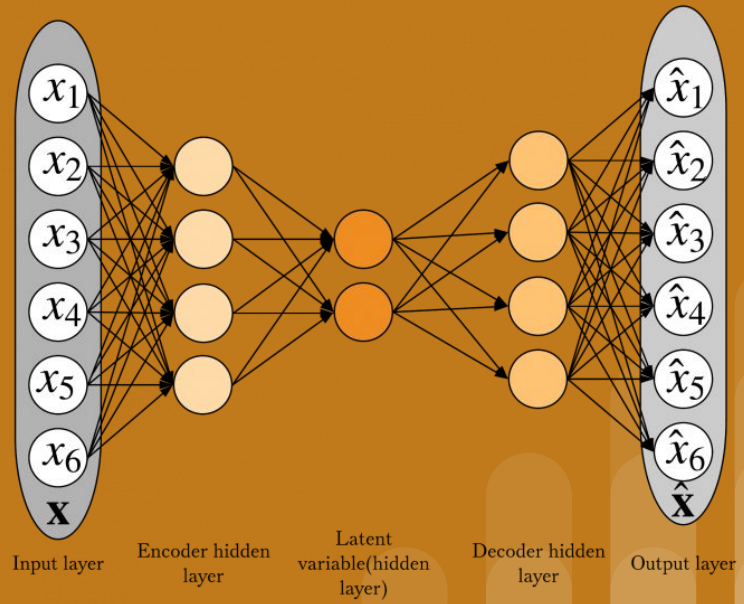
Si me paro en (0, 0,5)

¿Cómo saber si es válido? Pareciera que no.. (Ej 2)

Denoiser

Denoiser Autoencoder - Optimización

- Autoencoder
- Input de entrada con ruido
- Espacio latente dimensión variable
- Variación de capas intermedias
- ruidos utilizados 10%, 15%, 20%



Denoiser Autoencoder - Optimización

Error en inputs sin ruido - 15.000 epochs

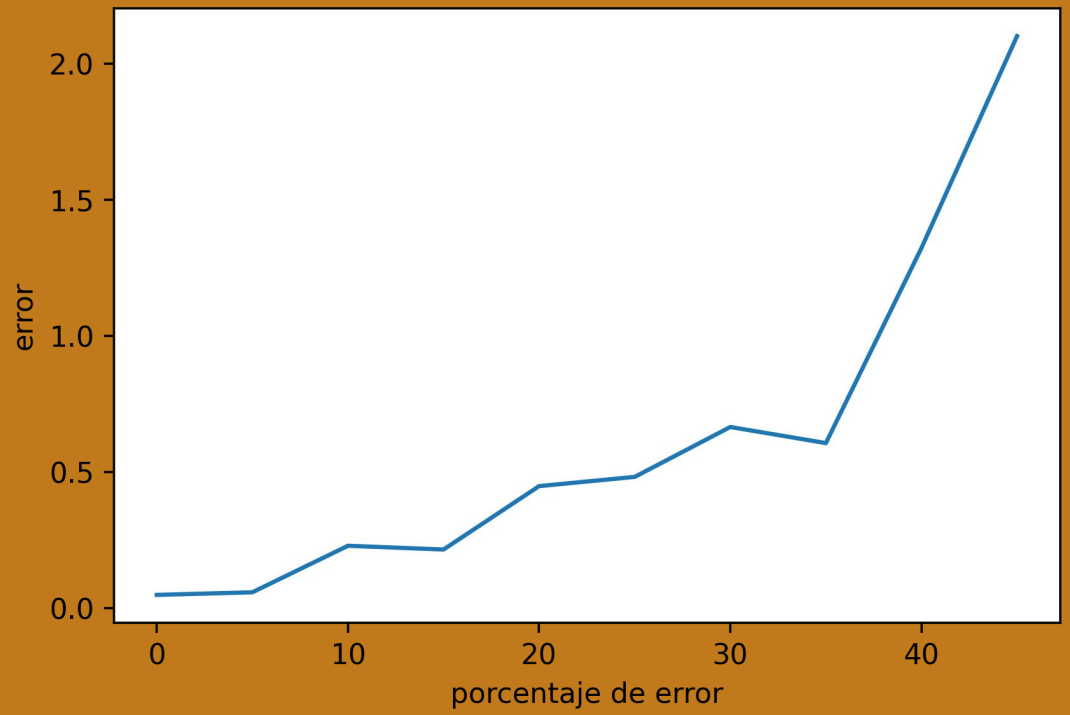
Distribución capas internas	Entrenado con ruido 10%	Entrenado con ruido 15%	Entrenado con ruido 20%
{64,32,16,2,16,32,64}	0.431	0.849	1.855
{32,16,32}	0.192	0.112	0.263
{32,16,8,16,32}	0.025	0.058	0.213
{64,32,16,32,64}	0.009	0.024	0.097

Denoiser Autoencoder - Optimización

15.000 epochs

porcentaje de ruidos usados para entrenar	Error input ruido 0%	Error input ruido 10%	Error input ruido 15%	Error input ruido 20%
[20]	0.0598	0.104	0.181	0.273
[10, 15, 20]	0.023	0.041	0.0765	0.302
[0, 10, 15, 20]	0.018	0.0272	0.0727	0.253

Denoiser Autoencoder - Optimización

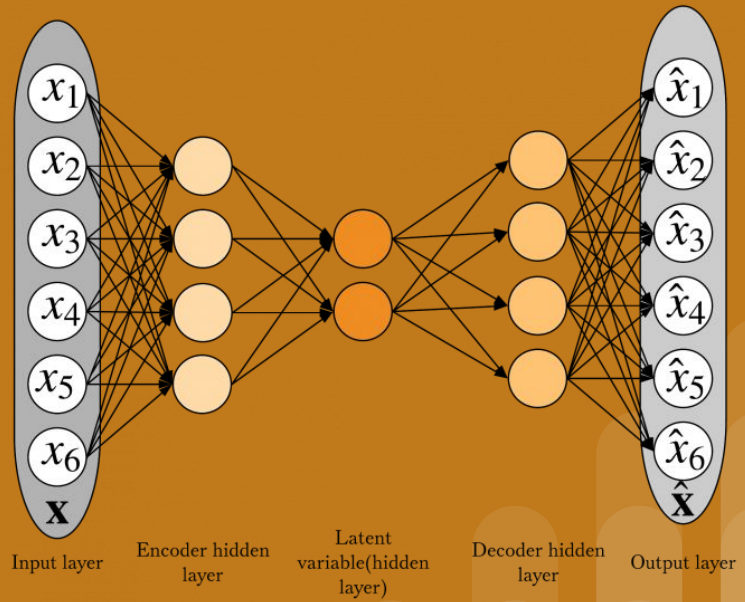


15.000 epochs

Distribución capas internas:
{64,32,16,32,64}

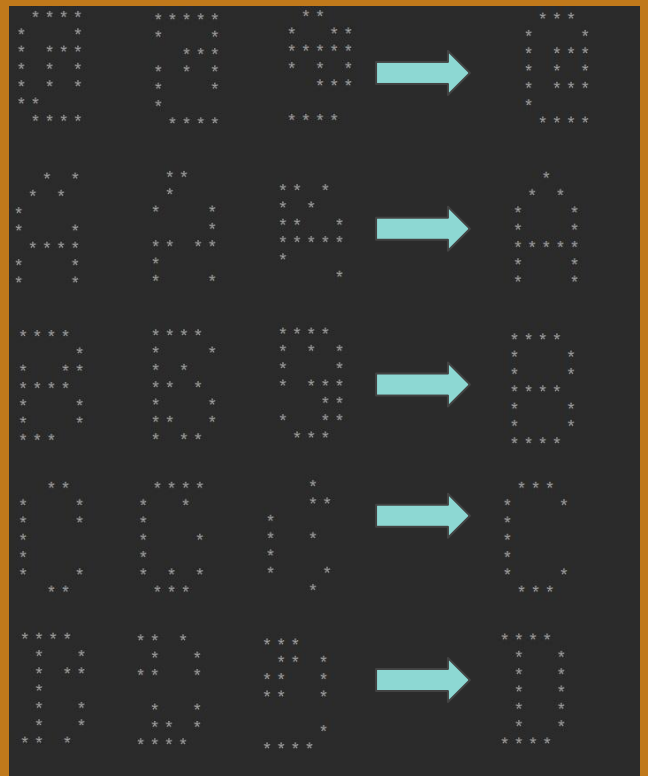
Denoiser Autoencoder - Ejemplo

- Función de activación TanH
- 40000 Epochs
- Distribución de capas internas {64, 32, 16, 32, 64}
- Momentum término 0.7
- Learning rate adaptativo comenzando en 0.001

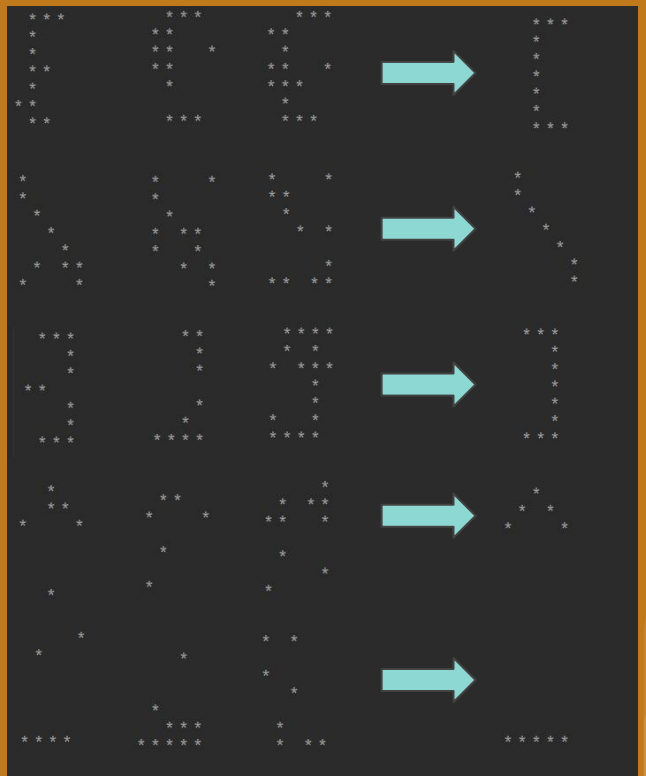


Denoiser Autoencoder - Ejemplo

10% 15% 20%



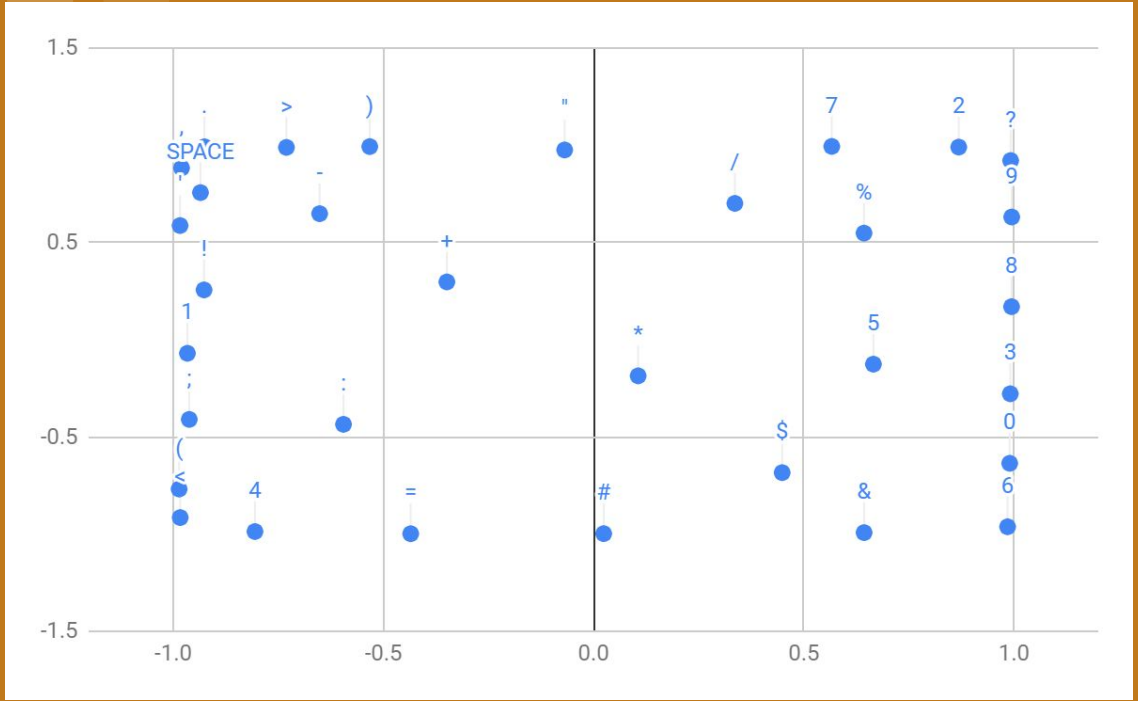
10% 15% 20%

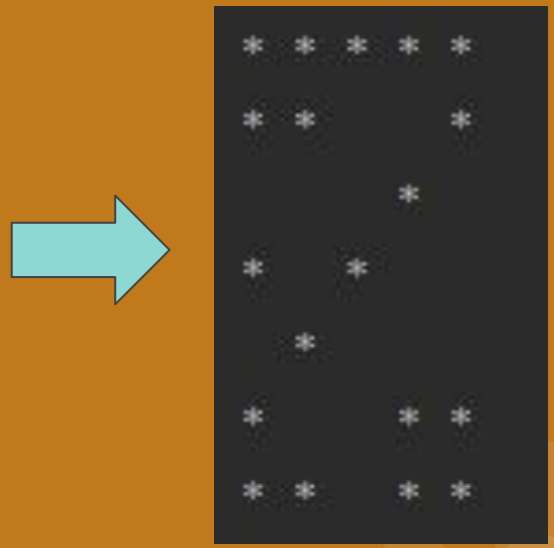


error 10% : 0.006
error 15% : 0.022
error 20% : 0.061

Capacidad generativa

Autoencoder - Generando nueva letra





¿Cómo recorro el espacio latente?

Capacidad generativa

- Definimos un set de “emojis” dibujando con bits binarios
- Un emoji que “pertenezca” al set debería tener dos ojos y una boca definidos

```
int[][] font1= {{0x00, 0x0A, 0x0A, 0x00, 0x11, 0x0E, 0x00},   // :
       {0x00, 0x0A, 0x0A, 0x00, 0x1F, 0x11, 0x0E},  // :D
       {0x00, 0x0A, 0x0A, 0x00, 0x0E, 0x11, 0x00}, // :(
       {0x00, 0x0A, 0x0A, 0x00, 0x0E, 0x11, 0x1F}, // D:
       {0x00, 0x0A, 0x0A, 0x00, 0x00, 0x1F, 0x00}, // :|
       {0x00, 0x0A, 0x0A, 0x00, 0x11, 0x1F, 0x11}, // :I
       {0x00, 0x0A, 0x0A, 0x00, 0x1F, 0x05, 0x02}, // :P
       {0x00, 0x0A, 0x0A, 0x00, 0x0E, 0x11, 0x0E}, // :O
       {0x00, 0x08, 0x0B, 0x00, 0x11, 0x0E, 0x00}, // ;)
       {0x00, 0x08, 0x0A, 0x00, 0x1F, 0x15, 0x0A},  // :B
       {0x00, 0x08, 0x0A, 0x00, 0x0A, 0x04, 0x00},   // c:
       {0x00, 0x08, 0x0A, 0x00, 0x04, 0x0A, 0x00},   // :c
       {0x00, 0x08, 0x0A, 0x00, 0x15, 0x0A, 0x00},   // :3
       {0x00, 0x08, 0x0A, 0x00, 0x0A, 0x15, 0x00},   // 3:
       {0x00, 0x08, 0x0A, 0x00, 0x11, 0x1F, 0x00},   // :]
       {0x00, 0x08, 0x0A, 0x00, 0x1F, 0x11, 0x00},   // :[
       {0x00, 0x08, 0x0B, 0x00, 0x1F, 0x11, 0x0E},// ;D
```

Capacidad generativa

¿Qué ocurre si tomamos posiciones azarosas del espacio latente?

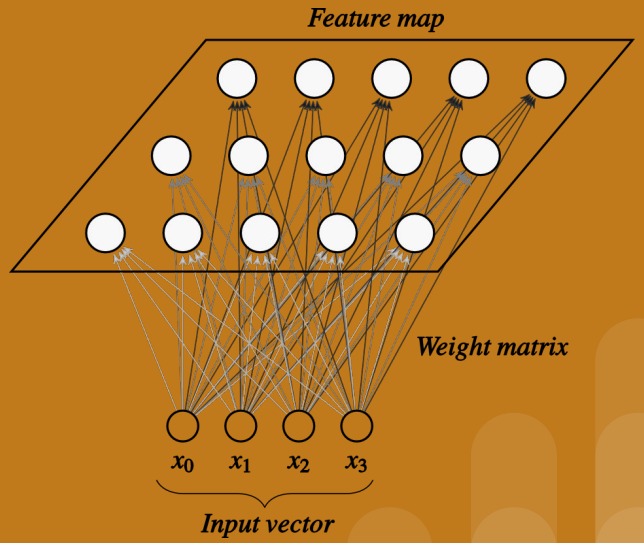
Tiene ojos y boca, podría pertenecer

No pertenece

Capacidad generativa

¿Cómo recorremos el espacio latente?

- Utilizamos como criterio una red de Kohonen
- La red describe y le da una estructura al espacio latente
- Si un elemento cae en una neurona existente, tendría la misma clasificación y por ende pertenecería al conjunto



Capacidad generativa

Ejemplos concretos

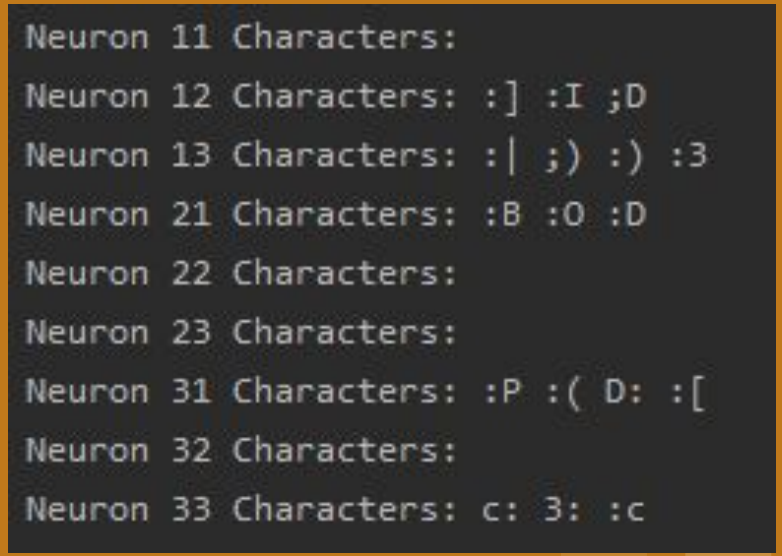
- Red de Kohonen 3x3
- Input: coordenadas de los elementos en el espacio latente
- Observamos la clasificación

```
Neuron 11 Characters:
Neuron 12 Characters: :] :I ;D
Neuron 13 Characters: :| ;) :) :3
Neuron 21 Characters: :B :O :D
Neuron 22 Characters:
Neuron 23 Characters:
Neuron 31 Characters: :P :( D: :[
Neuron 32 Characters:
Neuron 33 Characters: c: 3: :c
```

Capacidad generativa

Ejemplos concretos

:] y ;D caen en la misma neurona. Nos movemos en una recta entre ellos en el espacio latente.

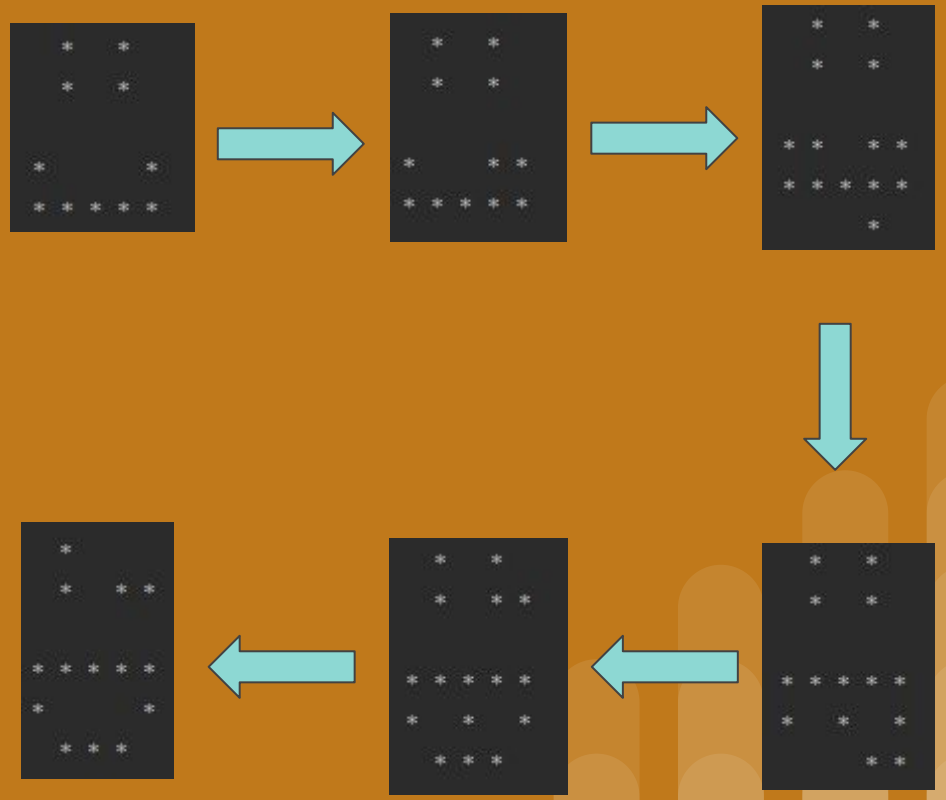


Capacidad generativa

Ejemplos concretos

:] y ;D caen en la misma neurona. Nos movemos en una recta entre ellos en el espacio latente.

Mantienen las características propias del conjunto de datos: ojos y boca.

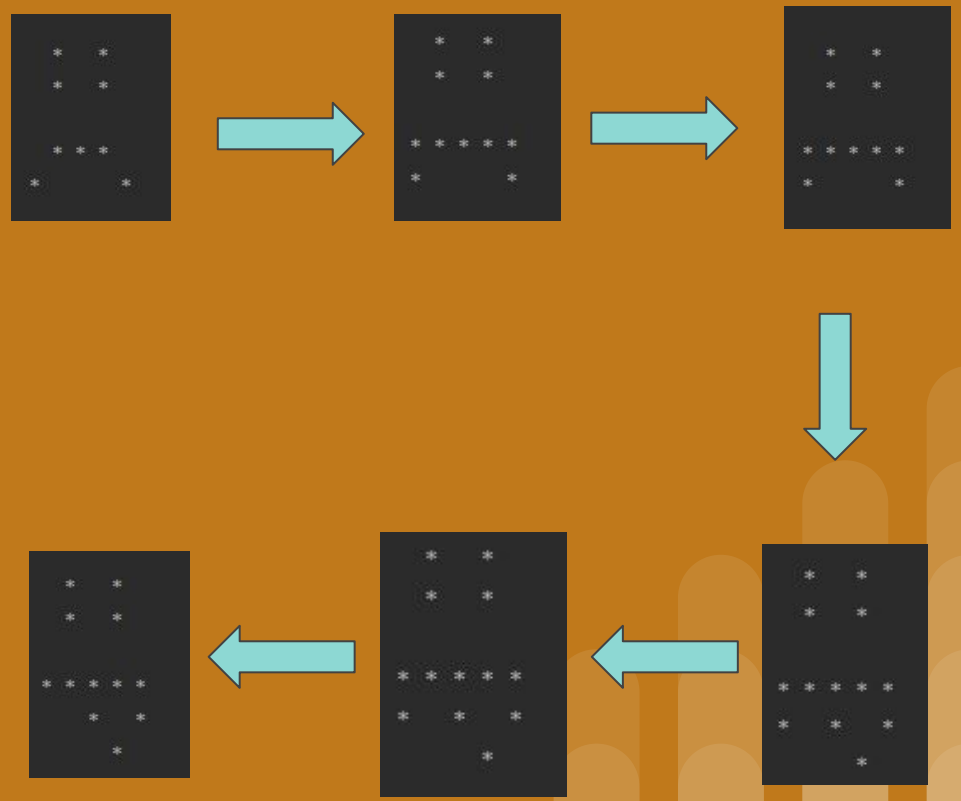


Capacidad generativa

Ejemplos concretos

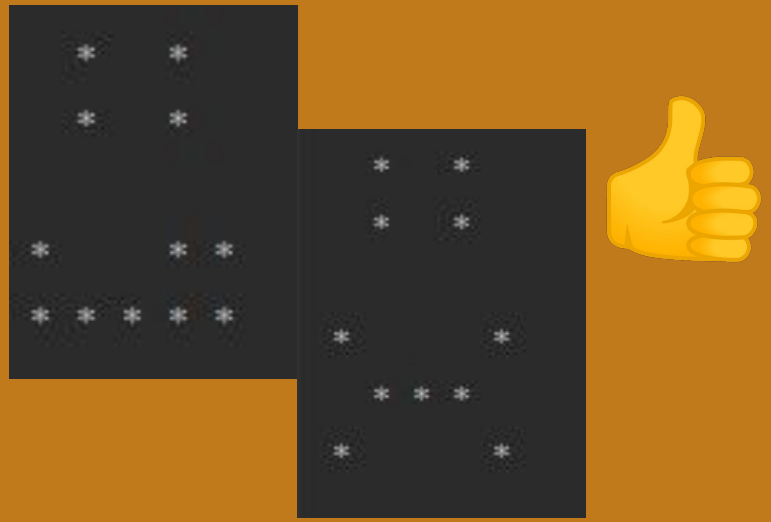
Probemos con :P y :(

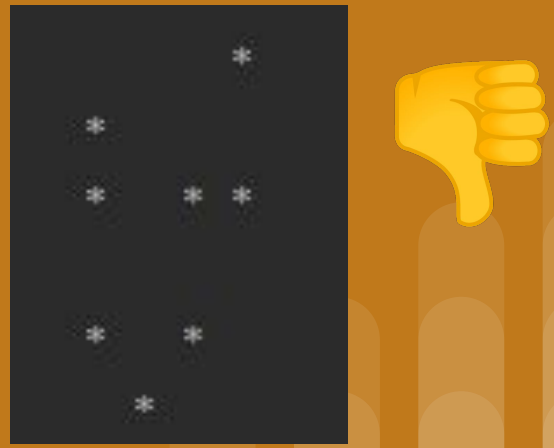
Mantienen las características propias del conjunto de datos: ojos y boca.



Capacidad generativa

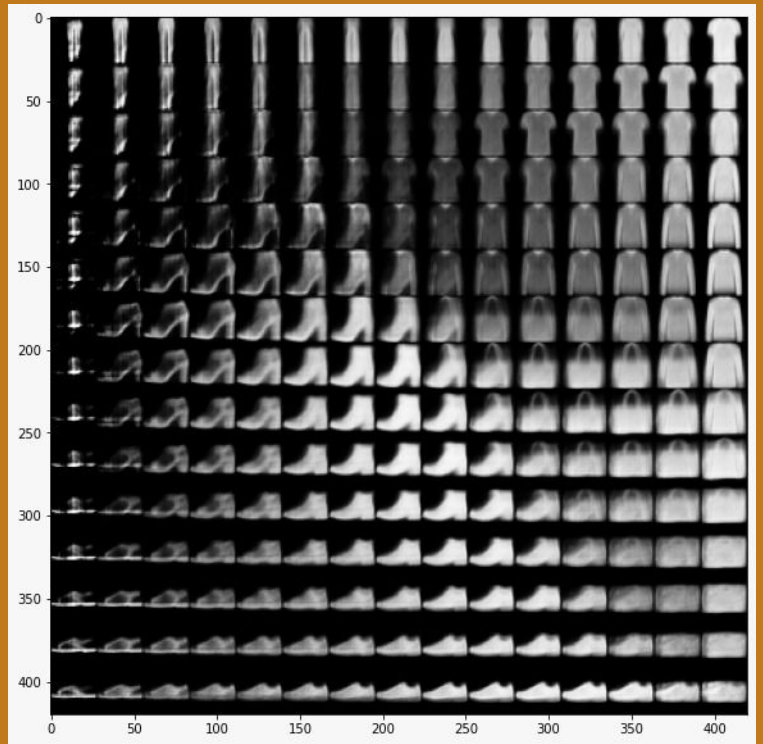
Manteniendo este criterio, podríamos movernos según los pesos de Kohonen y obtener muestras que se adecúan al conjunto de datos.

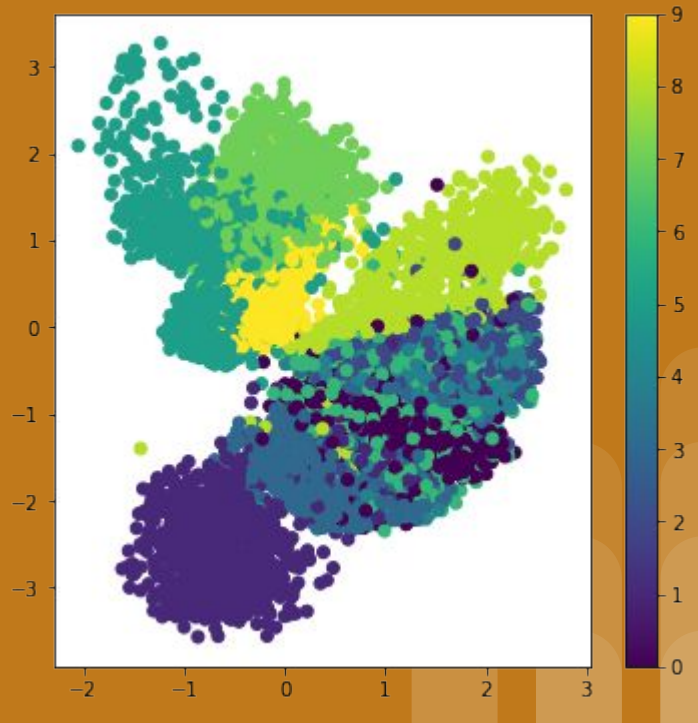




Capacidad generativa

Comparación con Autoencoder Variacional (Keras)





Conclusiones

Conclusiones

- Existen varias maneras de optimizar el Autoencoder
- No hay una fórmula fija, se deben realizar experimentaciones
- Hay que adaptar la estructura al problema presentado
- Combinar distintos métodos de optimización no siempre da el mejor resultado
- El Denoising Autoencoder hace un buen trabajo eliminando inputs con ruido
- Se debe definir una estructura del espacio latente para poder recorrerlo apropiadamente y poder aprovechar su capacidad generativa
- Se puede abarcar el problema de la reestructuración de distintas maneras