



SIA TP5

Gastón Lifschitz
Lucio Pagni

Fonts.java



{0x04, 0x0a, 0x11, 0x11, 0x1f, 0x11, 0x11}, // 0x41, A



00100
01010
10001
10001
11111
10001
10001



0.0 0.0 1.0 0.0 0.0 0.0 1.0 0.0 1.0 0.0 1.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0 1.0 1.0 1.0 1.0 1.0 0.0 0.0 0.0 1.0 1.0 0.0 0.0 0.0 1.0



Autoencoders

Implementar un Autoencoder básico



Autoencoder

- int nInputs
- int nHidden
- int nEpochs
- Neuron[] hiddenPerceptrons
- Neuron[] inputPerceptrons
- Neuron[] output
- double learning_rate
- Neuron[] nHiddenL
- Neuron[] nHiddenR
- double[] minNeuronHidden
- double[] maxNeuronHidden

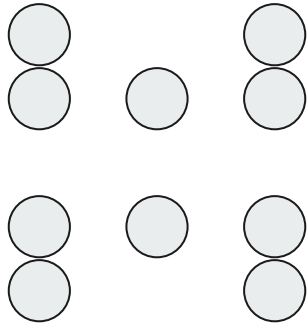
- Neuron[] getOutput()
- Autoencoder(int nInputs, int nHidden, int nEpochs, double learning_rate, double beta)
- void train(double[] input)
- double dotProduct(double[] weights, Neuron[] perceptrons)
- double dotProduct(double[] weights, double[] perceptrons)
- void updateError(double[] expectedResult)
- void printArray(double[] input)
- boolean run(double[] input)
- String espacioLatente()
- void difference(double[] a, double[] b)
- ▲ void write(String filename, String value)
- double[] normalize(double[] matrix)
- double[] generateRandom()

Arquitectura

La arquitectura de la red consiste en



35 nodos de entrada 10 / 32 / 35
nodos en la capa oculta y 35 nodos en
la capa de salida.



Nodos en la capa de entrada

```
Autoencoder autoencoder = new Autoencoder(35, 10, 1000, 0.2, 0.1);
```

Nodos en la capa de oculta

```
Autoencoder autoencoder = new Autoencoder(35, 35, 1000, 2, 2.5);
```

```
Autoencoder autoencoder = new Autoencoder(35, 35, 1000, 2, 2.5);
```

Optimizaciones

Hacemos la resta entre el
valor esperado y la salida
obtenida

Multiplicamos el valor anterior por la derivada

Método General de Optimización

```
for (int i = 0; i < output.length; i++) {  
    output[i].setError((expectedResult[i] - output[i].getOutput()) * (output[i].derivative()));  
}
```

El algoritmo de optimización utilizado es el método del gradiente descendiente donde la dirección de descenso está dada por el método del gradiente descendiente.

- Se intentó implementar el método de sacudida pero no logra converger (perjudicaba al algoritmo). El método hacía que 1 de cada 10 (aprox) actualizaciones de pesos se le agregaba un 0.1 al valor del peso elegido aleatoriamente.

Optimizaciones

```
Autoencoder autoencoder = new Autoencoder(35, 35, 1000, 2, 2.5);
```

```
public Autoencoder(int nInputs, int nHidden, int nEpochs, double learning_rate, double beta){
```

beta = 2.5



Estudiamos la evolución según el learning rate

Correctos	Training Set	Time Spent	Learning Rate
32	32	12867	2
32	32	4671	1.5
32	32	15178	1
32	32	1770	0.5
32	32	2842	0.25
32	32	6806	0.125

Optimizaciones

```
Autoencoder autoencoder = new Autoencoder(35, 35, 1000, 2, 2.5);
```

```
public Autoencoder(int nInputs, int nHidden, int nEpochs, double learning_rate, double beta){
```

learning rate = 2

Estudiamos la evolución según beta

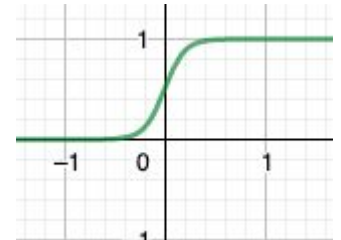
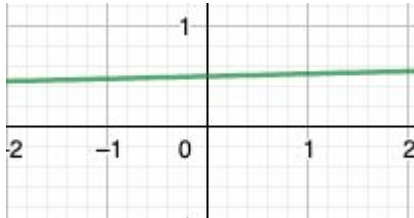
Correctos	Training Set	Time Spent	Beta
32	32	1774	2.5
32	32	2292	2
32	32	8856	1.5
32	32	8578	1
32	32	40385	0.5
32	32	10546	0.25

Optimizaciones

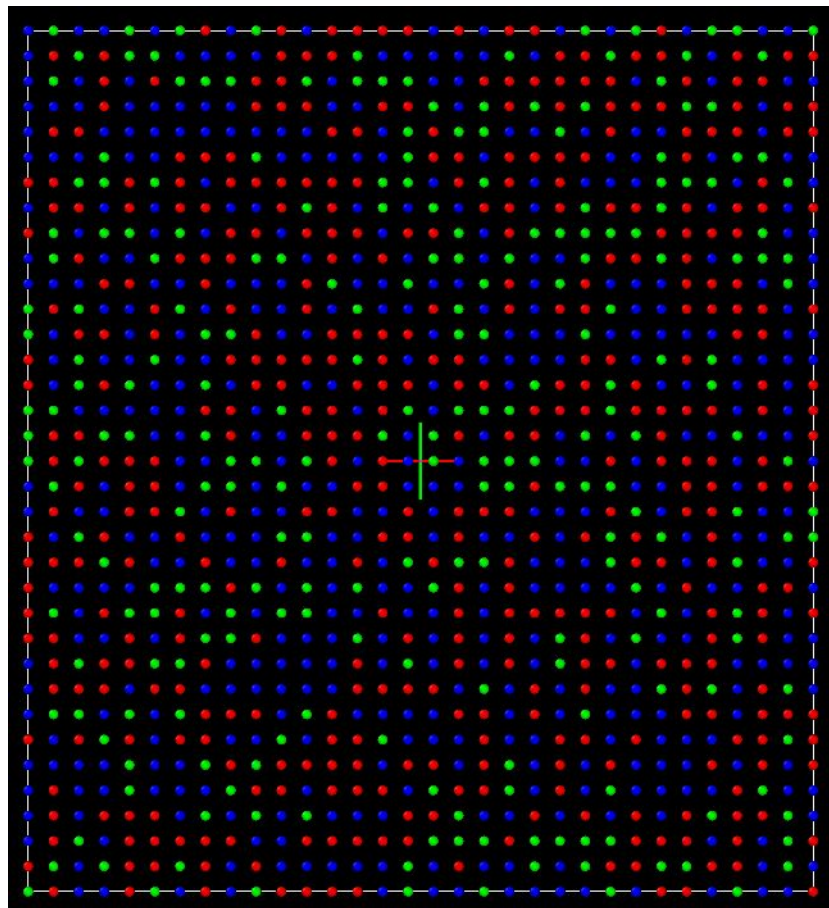
El parámetro beta regula que tanto la función de activación se parece a una función lineal o a una función escalón

Cuando beta tiende a cero la función de activación tiende a ser una función lineal

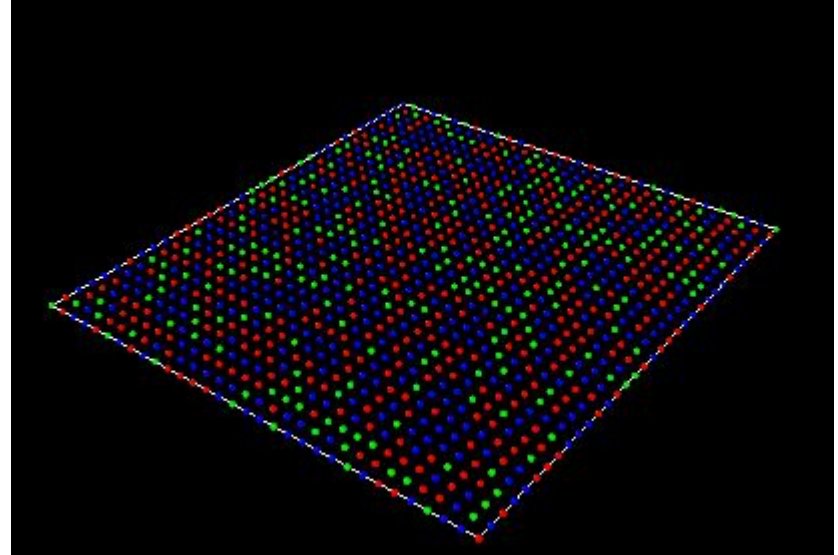
A medida que se baja el beta el tiempo de entrenamiento se incrementa



Los datos de entrada en el espacio latente



Evolución del espacio latente en el tiempo



Evolución de los pesos en el tiempo con logística

Evaluación de los pesos en el Tiempo con tanh

Resultados - 35x35x35

Con Sigmoide



Set de entrenamiento	Tiempo	Aprende todos?
28	170 segundos	Si
29	190 segundos	Si
30	210 segundos	Si
31	315 segundos	Si
32	Infinito	No

Resultados - 35x32x35

Con Sigmoide



Set de entrenamiento	Tiempo	Aprende todos?
27	32 segundos	Si
28	55 segundos	Si
29	97 segundos	Si
30	104 segundos	Si
31	Infinito	No
32	Infinito	No



Ahora veamos con Tanh....

Tanh

- Se logra aprender todos los patrones dependiendo la arquitectura
- Error menor que 0.01!!!! Antes con logistica lo haciamos con menor que 0.09

Capa oculta	Learning rate	Beta	Tiempo	Pasos
35	0.5	0.5	17 segundos	2500
30	0.5	0.5	15 segundos	3250
25	0.5	0.5	19 segundos	2426
20	0.5	0.5	25 segundos	4502
15	0.5	0.5	24 segundos	4003

Generación de nuevas letras



```
Random r = new Random();
double[] outputVal = new double[output.length];
double[] randomHidden = new double[minNeuronHidden.length];
for(int i = 0; i < minNeuronHidden.length; i++){
    randomHidden[i] = minNeuronHidden[i] + (maxNeuronHidden[i] - minNeuronHidden[i]) * r.nextDouble();
}
for(int i = 0; i < output.length; i++){
    double sumOfWeights = dotProduct(output[i].getWeights(), randomHidden);
    outputVal[i] = output[i].activationFunction(sumOfWeights);
}
return outputVal;
```




Denoising Autoencoders

Tienen la capacidad de poder reconstruir datos corrompidos

Si la capa oculta tiene más nodos que la capa de entrada se corre el riesgo de caer en la función identidad o función nula, es decir que la entrada es igual a la salida.

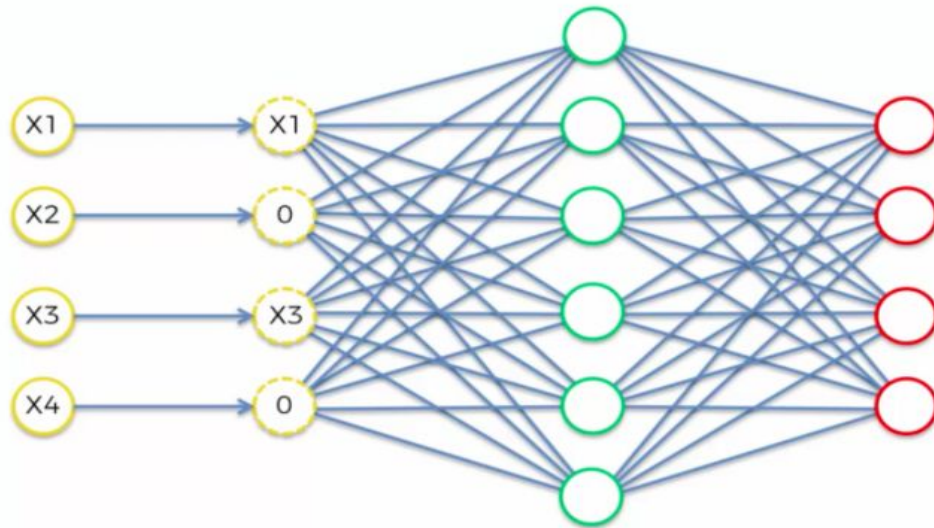
Para resolver este problema surgen los denoising autoencoders.

Esto se hace inicializado en cero valores al azar.

Plantear una arquitectura de red conveniente

Vemos que ciertos valores de la entrada se hacen cero de manera aleatoria.

Esto es lo que se conoce como ruido.



Implementar un Denoising Autoencoder



DenoisingAutoencoder

```
□ int nInputs
□ int nHidden
□ int nEpochs
□ Neuron[] hiddenPerceptrons
□ Neuron[] inputPerceptrons
□ Neuron[] output
□ double learning_rate
○ double[] diff
○ double[] out

● Neuron[] getOutput()
● DenoisingAutoencoder(int nInputs, int nHidden, int nEpochs, double learning_rate, double beta)
● void train(double[] input)
■ double dotProduct(double[] weights, Neuron[] perceptrons)
● void updateError(double[] expectedResult)
■ void printArray(double[] input)
● boolean run(double[] input)
● String espacioLatente()
■ void difference(double[] a, double[] b)
▲ void write(String filename, String value)
```

```
for (int i = 0; i < TRAINING_SET; i++) {  
    autoencoder.train(fonts.print(i));  
    autoencoder.updateError(fonts.print(i));  
}
```

```
for (int i = 0; i < TRAINING_SET; i++) {  
    autoencoder.train(fonts.printWithNoise(i));  
    autoencoder.updateError(fonts.print(i));  
}
```

El proceso de entrenamiento del Autoencoder se hace según los patrones de entrada en fonts.

El proceso de entrenamiento del Denoising Autoencoder se hace según los patrones de entrada en fonts con cierto nivel de ruido, el cual es autogenerado y parametrizable.

Estos patrones distorsionados se usan como entrada pero se conserva el patrón original en la salida.

Esto se hace para entrenar la red.

Una vez entrenada se puede tomar un patrón al azar, distorsionarlo, ponerlo en la red.

Se observa que la red recupera el valor original, es decir aprende a eliminar el ruido.

Plantear una arquitectura de red conveniente

```
DenoisingAutoencoder autoencoder = new DenoisingAutoencoder(35, 35, 1000, 2, 3);
```

```
DenoisingAutoencoder autoencoder = new DenoisingAutoencoder(35, 32, 1000, 2, 3);
```

```
DenoisingAutoencoder autoencoder = new DenoisingAutoencoder(35, 10, 1000, 2, 3);
```

Experimentamos con varias arquitecturas de red -----> Ver diapo. 15

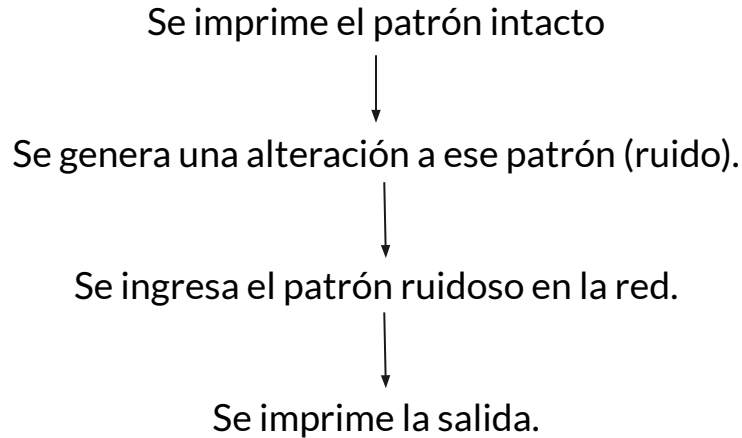
35xNx35 -> El entrenamiento es 'rápido'. N en [15;35]

35x10x35 -> El entrenamiento es extremadamente lento.

Estudie la capacidad del autoencoder de eliminar ruido con entradas distorsionadas

```
for (int i = 0; i < TRAINING_SET; i++) {  
    autoencoder.train(fonts.printWithNoise(i));  
    autoencoder.updateError(fonts.print(i));  
}
```

```
System.out.println("Denoiser");  
double [] normal = fonts.print(i);  
printMatrix(normal);  
double [] ruido = fonts.printWithNoise(i);  
System.out.println("Ruido");  
printMatrix(ruido);  
boolean ret = autoencoder.run(ruido);  
  
//printArray(autoencoder.diff);  
printArray(autoencoder.out);  
  
//System.out.println(ret);
```



Denoiser

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

Ruido

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

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

Denoiser

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

Ruido

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

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

[Estudio en detalle](#)

Utilizar el autoencoder para generación de muestras






Se utilizó el autoencoder para generar muestras.

[Patrones Generados](#)



Explotando la capacidad generativa del Autoencoder

Dataset Shapes

	Shapes
	<code>double[][] Shapes</code>
	<code>double[] getShape(int i)</code>

- Formas como triángulos y rectángulos
- Size = 11
- Arquitectura 25xNx25
- Con una arquitectura 25x10x25 -> Aprende todos los patrones en 1-5 segundos aprox. (beta = 2.5)
- Inclusive se llega a una solución con arquitectura 25x6x25 entre 5-15 segundos (en algunos casos mayor)
- Tiene una matriz de double y un getter que obtiene a un array según su índice.
- Nuestro objetivo es a partir de este set de datos construido por nosotros , usar el Autoencoder para generar nuevos patrones los cuales nosotros podemos clasificar si pertenecen o no al conjunto.

Generación de muestras

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

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

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

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

```
Autoencoder autoencoder = new Autoencoder(25, 10, 1000, 2, 2.5);
```

Primero se aprende entre qué valores pueden ser las salidas de los perceptrones de la capa oculta.

Se generan números aleatorios entre esos rangos para representar posibles salidas.

Una vez generadas estos posibles valores se hacen pasar a la capa de salida para generar las imágenes.



Conclusiones

Conclusiones



Los autoencoders tienen la capacidad de reconstruir datos corrompidos. Puede verse como una forma de hacer PCA.

Un denoising autoencoder tiene la capacidad de eliminar el ruido de patrones ruidosos.

Para explotar la capacidad generativa del Autoencoder debemos aprender la salida de los perceptrones de la capa oculta.

Se buscó la optimización vía métodos de optimización y vía búsqueda de parámetros tales que se optimice el proceso de entrenamiento del autoencoder.

La función de activación tangente hiperbólica es una mejora sobre la función logística

La función ReLu se probó pero ya que los pesos suelen ser muy negativos, no lograba converger (peor que logística).