

Práctica 2. Redes neuronales

Sistemas Inteligentes – Ester
Martínez Martín
Curso 2019/2020

Objetivo

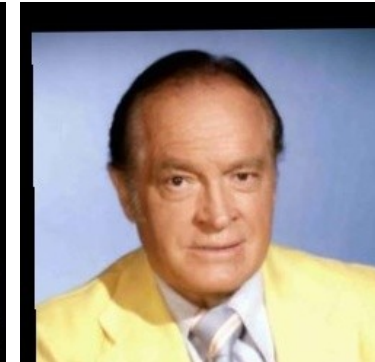
Diseñar una red neuronal
que determine el sexo de
las personas que aparecen
en una imagen

Qué vamos a usar

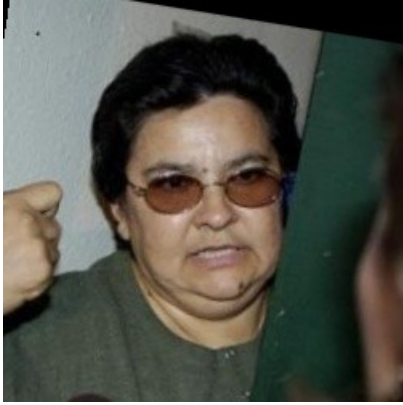


Cómo lo vamos a hacer

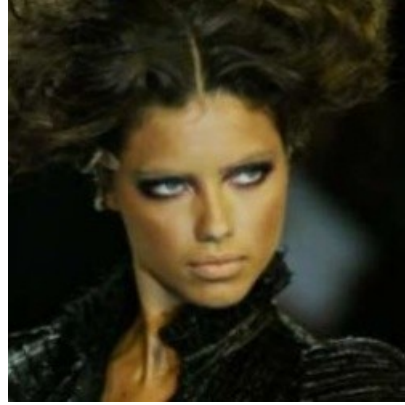
Labeled
Faces
in the Wild



Cómo lo vamos a hacer



female



female



female



female



Cómo lo vamos a hacer

female



female



female



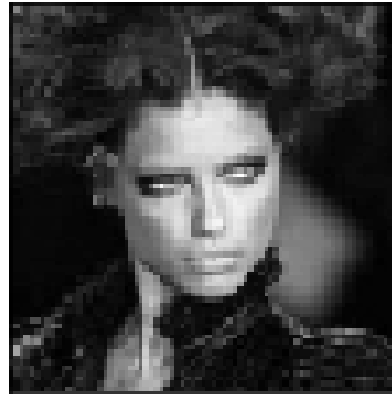
female



female



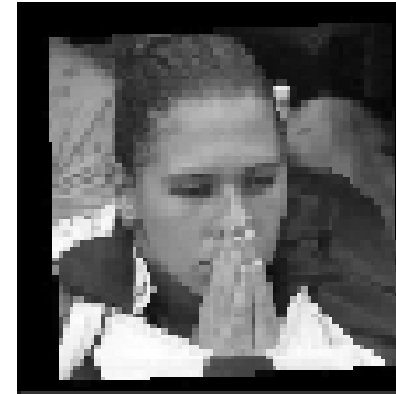
female



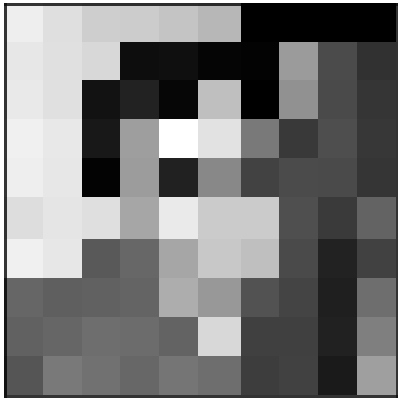
female



female



Cómo lo vamos a hacer



171	159	146	145	139	130	0	0	0	0
165	160	154	8	9	3	2	108	50	33
167	160	11	22	4	135	0	101	49	35
172	166	15	110	184	161	83	37	52	36
171	165	2	109	21	94	44	50	49	35
158	164	159	116	168	144	144	53	38	67
172	165	60	70	116	142	135	49	22	43
69	64	66	68	122	107	55	45	20	75
66	69	75	74	67	154	42	42	21	87
57	83	77	70	80	75	40	43	16	111

171
159
146
145
139
130
0
0
0
0
165
160
154
8
9
3
...
40
43
16
111

Número de
neuronas
en la capa de
entrada
=
Número de
píxeles
en la imagen

Cómo lo vamos a hacer



Cómo lo vamos a hacer

Clase DataSet

```
// Creación de un objeto de la clase DataSet
DataSet dataset;
// Indicamos el nombre del fichero que
contiene nuestro dataset

dataset.set_data_file_name("nuestraDataset.d
at");
// Indicamos cómo se han separado los datos
```

Cómo lo vamos a hacer

Clase Dataset - Modificar atributos

```
Variables* variables_pointer =  
dataset.get_variables_pointer();  
variables_pointer->set_name(index_Column,  
"NameAttribute");  
variables_pointer->set_units(index_Column,  
"AttributeUnit"); // Ej. "centimeters"  
variables_pointer->set_use(index_Column,  
AttributeUse); // Variables::Input - Variables::Target
```

Cómo lo vamos a hacer

Clase Dataset - Visualizar atributos

```
const Matrix<string> inputs_information = variables_pointer->get_inputs_information();  
const Matrix<string> targets_information = variables_pointer->get_targets_information();
```

```
cout << "Input information" << endl << inputs_information << endl;  
cout << "Target information" << endl << targets_information << endl;
```

Cómo lo vamos a hacer

Clase Dataset - Dividir datos

```
Instances* instances_pointer =  
dataset.get_instances_pointer();
```

```
// %training, %selection, %test  
instances_pointer->split_random_indices(0.7, 0.15, 0.15);
```

Cómo lo vamos a hacer

Clase NeuralNetwork

// Crea una red neuronal con un perceptron de 2 capas

```
NeuralNetwork neural_network(num_input, num_neuronas_oculta,  
num_output);
```

// Indicamos las entradas y las salidas

```
Inputs* inputs_pointer = neural_network.get_inputs_pointer();  
inputs_pointer->set_information(inputs_information);  
Outputs* outputs_pointer = neural_network.get_outputs_pointer();  
outputs_pointer->set_information(targets_information);
```

Cómo lo vamos a hacer

Clase NeuralNetwork

```
NeuralNetwork neural_network;
```

```
MultilayerPerceptron multLayer;
```

```
Vector<int> initialiceMultPercept;
```

```
initialiceMultPercept.push_back(num_input);
```

```
initialiceMultPercept.push_back(num_neuronas_ocultas_capaN);
```

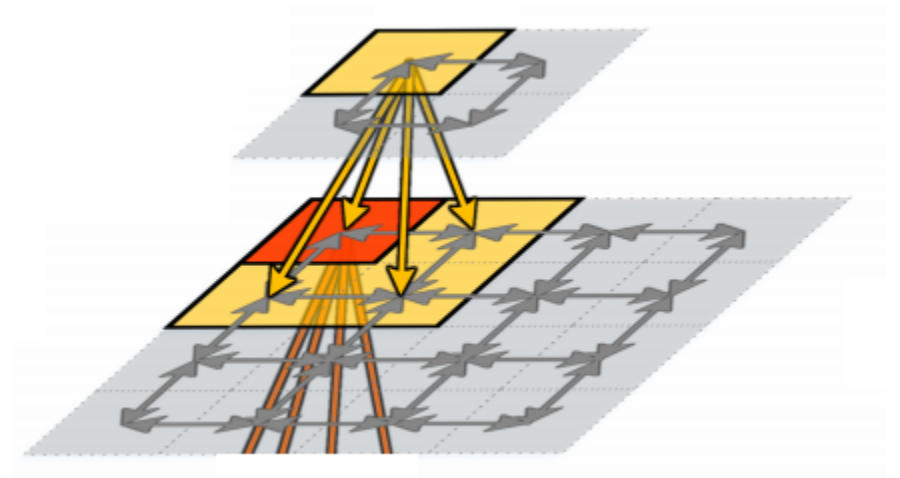
```
initialiceMultPercept.push_back(num_ouput);
```

```
neural_network.set(multLayer);
```

Cómo lo vamos a hacer

Clase NeuralNetwork

```
neural_network.construct_scaling_layer();  
neural_network.construct_unscaling_layer();
```

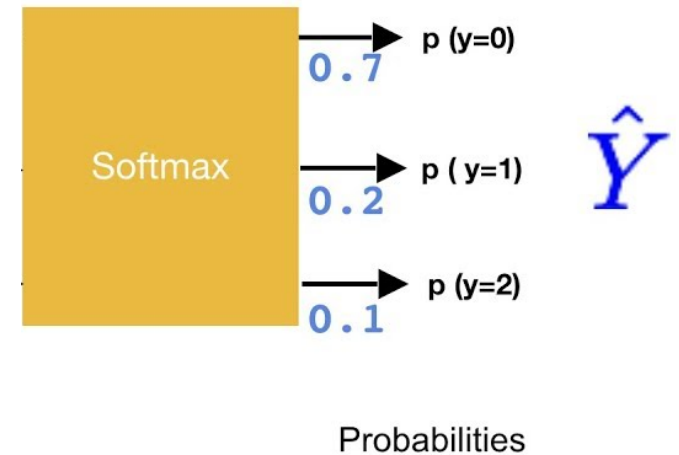


Cómo lo vamos a hacer

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^K e^{z_k}} \quad \text{for } j = 1, \dots, K.$$

Clase NeuralNetwork

```
// Para el reconocimiento de patrones
neural_network.construct_probabilistic_layer();
ProbabilisticLayer* probabilistic_layer_pointer =
neural_network.get_probabilistic_layer_pointer();
probabilistic_layer_pointer-
>set_probabilistic_method(ProbabilisticLayer::Softmax);
```



Cómo lo vamos a hacer

Training Strategy

- Estrategia de entrenamiento
 - Se encarga del entrenamiento de la red
 - Está compuesta de dos clases:
 - LossIndex: tipo de error
 - TrainingAlgorithm: tipo de entrenamiento
 - La elección del tipo de error y el de entrenamiento dependen de la aplicación

Cómo lo vamos a hacer

Training Strategy

```
TrainingStrategy training_strategy(&neural_network,  
&dataset);
```

- Por defecto:

LossIndex – NORMALIZED_SQUARED_ERROR

TrainingAlgorithm – QUASI_NEWTON_METHOD

Cómo lo vamos a hacer

Training Strategy

```
TrainingStrategy training_strategy(&neural_network,  
&dataset);
```

- **Para cambiarlos:**

```
TipoESTRATEGIA* ptro =  
training_strategy.get_ESTRATEGIA_pointer();  
QuasiNewtonMethod* quasi_Newton_method_pointer =  
training_strategy.get_quasi_Newton_method_pointer();
```

Cómo lo vamos a hacer

Training Algorithm - Opciones

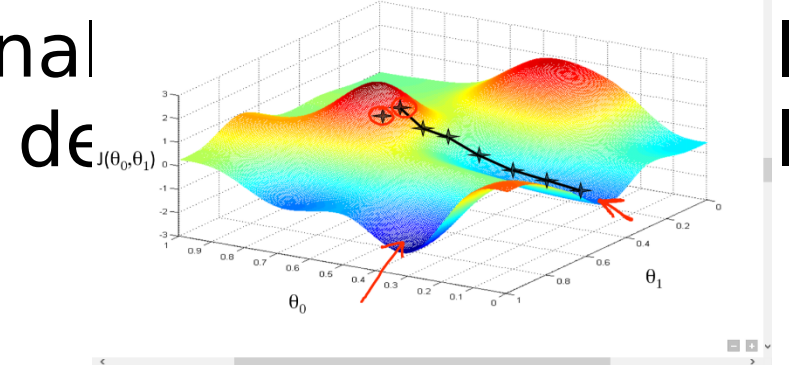
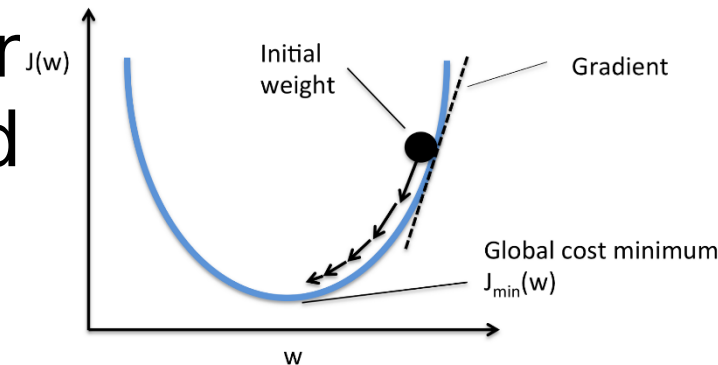
- GRADIENT_DESCENT
- CONJUGATE_GRADIENT
- NEWTON_METHOD
- QUASI_NEWTON_METHOD
- LEVENBERG_MARQUARDT_ALGORITHM

Cómo lo vamos a hacer

Training Algorithm -

Gradient_Descent

El descenso de gradiente es un algoritmo de optimización iterativo de primer orden para encontrar el mínimo de una función muy utilizado en Machine Learning. Para encontrar el mínimo local de una función que utiliza el descenso de gradiente, uno toma un punto inicial y se mueve en la dirección opuesta al gradiente (o gradiente negativo) hasta que se alcanza el mínimo.

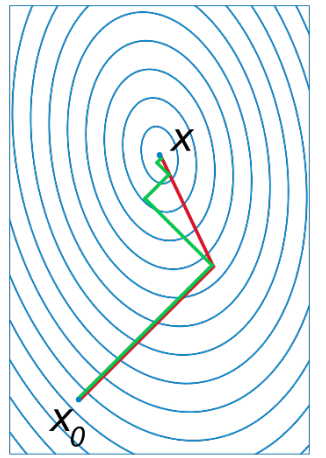


Cómo lo vamos a hacer

Training Algorithm -

Conjugate_gradient

El método de gradiente conjugado es un algoritmo para la solución numérica de sistemas particulares de ecuaciones lineales, es decir, aquellos cuya matriz es simétrica y positiva definida. El método de gradiente conjugado a menudo se implementa como un algoritmo iterativo, aplicable a sistemas dispersos que son demasiado grandes para ser manejados por una implementación directa u otros métodos directos como la descomposición de Cholesky. Los

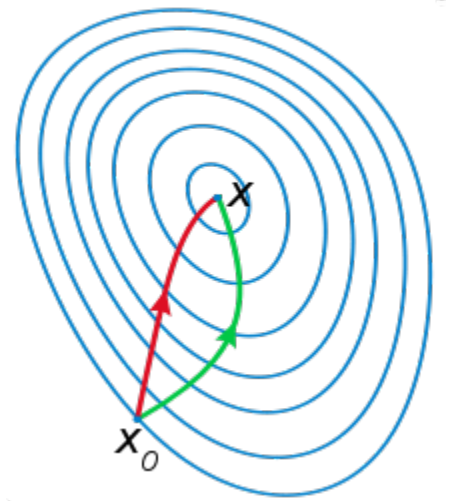


Cómo lo vamos a hacer

Training Algorithm -

Newton_method

El método de Newton es un método iterativo para encontrar las raíces de una función diferenciable f , que son soluciones a la ecuación $f(x) = 0$. Más específicamente, en la optimización, el método de Newton se aplica a la derivada f' de la función doble diferenciable f para encontrar las raíces de la derivada (soluciones a $f'(x) = 0$), también conocidas como los puntos estacionarios de f . Estas soluciones pueden ser mínimas, máximas o puntos de silla de montar.



Cómo lo vamos a hacer

Training Algorithm -

Quasi_Newton_method

Los métodos cuasi-Newton son métodos que se usan para encontrar ceros o máximos y mínimos locales de funciones, como una alternativa al método de Newton. Se pueden usar si el jacobiano o el hessiano no están disponibles o son demasiado caros para calcularlos en cada iteración. El método de Newton "completo" requiere que el jacobiano busque ceros, o la arpillera para encontrar los extremos.

Cómo lo vamos a hacer

Training Algorithm -

Levenberg-Marquardt_Algorithm

El algoritmo de Levenberg-Marquardt (LMA o simplemente LM) se utiliza para resolver problemas de mínimos cuadrados no lineales. Estos problemas de minimización surgen especialmente en el ajuste de curvas de mínimos cuadrados.

Cómo lo vamos a hacer

Loss Index - Opciones

- SUM_SQUARED_ERROR
- MEAN_SQUARED_ERROR
- ROOT_MEAN_SQUARED_ERROR
- NORMALIZED_SQUARED_ERROR
- MINKOWSKI_ERROR
- WEIGHTED_SQUARED_ERROR
- ROC_AREA_ERROR
- CROSS_ENTROPY_ERROR

Cómo lo vamos a hacer

Training Strategy

Para entrenar la red:

```
TrainingStrategy::Results results =  
training_strategy.perform_training();
```

Cómo lo vamos a hacer

Model Selection

La selección del modelo se aplica para encontrar una red neuronal con una topología que minimice el error para nuevos datos. Hay dos formas de obtener una topología óptima:

- la selección de orden: obtiene el

Cómo lo vamos a hacer

Model Selection

- Construcción del modelo de selección

ModelSelection

```
model_selection(&training_strategy);
```

- La selección de modelo por defecto consiste en un algoritmo de

Cómo lo vamos a hacer

Model Selection

- Construcción del modelo de selección

ModelSelection

```
model_selection(&training_strategy);
```

- Para cambiar la estrategia de selección:

Cómo lo vamos a hacer

Input Selection - Opciones

- GROWING_INPUTS
- PRUNING_INPUTS
- GENETIC_ALGORITHM

Cómo lo vamos a hacer

Order Selection - Opciones

- INCREMENTAL_ORDER
- GOLDEN_SECTION
- SIMULATED_ANNEALINGGENETIC_ALGORITHM

Cómo lo vamos a hacer

Model Selection

- Ejecutar el modelo de selección

// Selección de entradas

```
model_selection.perform_inputs_selection();
```

// Número de neuronas del perceptron

```
model_selection.perform_order_selection
```

Cómo lo vamos a hacer

Testing Analysis

El propósito de las pruebas es comparar los resultados de la red neuronal con los objetivos en un conjunto de pruebas independientes. Esto mostrará la calidad del modelo antes de su implementación.

Cómo lo vamos a hacer

Testing Analysis

Constructor

TestingAnalysis

```
testing_analysis(&neural_network, &dataset);
```

Problemas de clasificación – Regresión
lineal

```
Vector<TestingAnalysis::LinearRegressionAnaly
```

Cómo lo vamos a hacer

Testing Analysis

Constructor

TestingAnalysis

```
testing_analysis(&neural_network, &dataset);
```

Problemas de reconocimiento de
patrones – Matriz de confusión

```
Matrix<size_t> confusion_matrix =
```

Entrega

24 de mayo de 2020

Código fuente
Memoria