Sistemas Inteligentes para Gestión en la Empresa

Máster en Ingeniería Informática

04. Modelos avanzados – *Deep Learning*

UNIVERSIDAD
DE GRANADA

Deep Learning

Bibliografía

F. Chollet, J. Allaire (2018) Deep Learning with R. Manning.

F. Berzal (2018) Redes Neuronales & Deep Learning. https://sites.google.com/view/redes-neuronales/

I. Goodfellow, Y. Bengio, A. Courville (2016) *Deep Learning.* MIT Press. http://www.deeplearningbook.org

A. Trask (2018) *Grokking Deep Learning*. Manning.

M. Nielsen (2017) *Neural Networks & Deep Learning*. http://neuralnetworksanddeeplearning.com

DeepLearningAI: https://www.deeplearning.ai

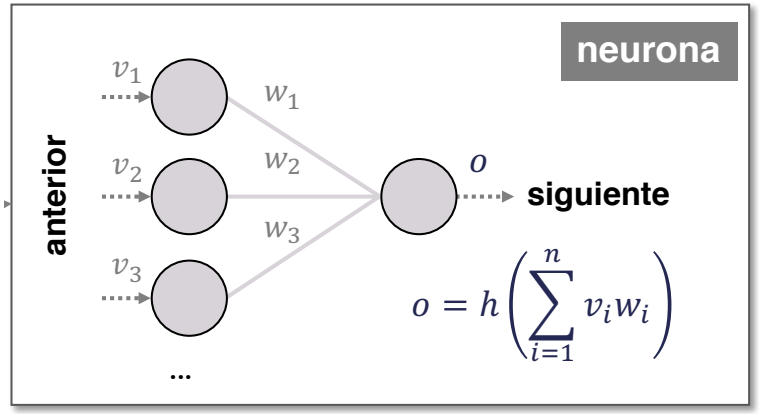
FastAI: https://course.fast.ai

F. Chollet (2017) Deep Learning with Python. Manning.

Índice

Deep Learning

Fundamentos



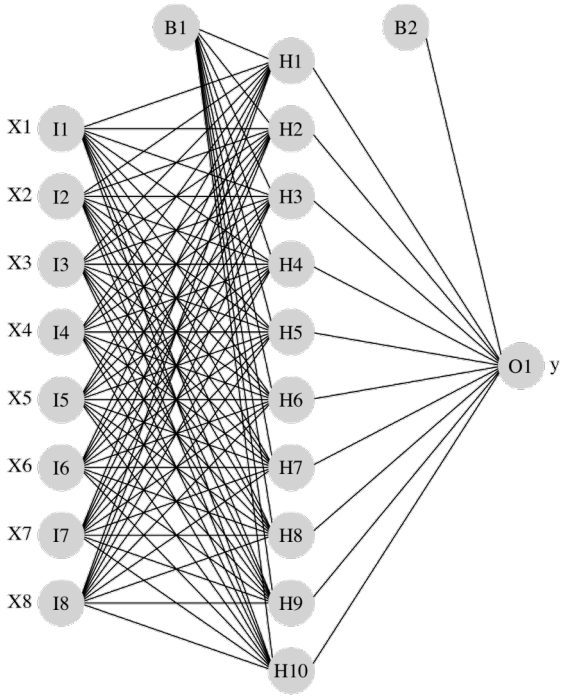
Carlos Santana Vega (DotCSV)

https://youtu.be/MRIv2IwFTPg

Partes 1, 2

Deep Learning

Fundamentos ▶ Elementos



perceptrón

Entrada (*n*)

$$x = (x_1, \ldots, x_n)$$

Salidas capa oculta (*m*)

$$H_j = h\left(\sum_{i=1}^{n} x_i w_{ij} + B_1\right)$$

dot product
producto escalar

$$x \cdot w = \begin{bmatrix} x_1 \\ \ldots \\ x_n \end{bmatrix} \begin{bmatrix} w_1 & \ldots & w_n \end{bmatrix}$$

Salidas finales (*k*)

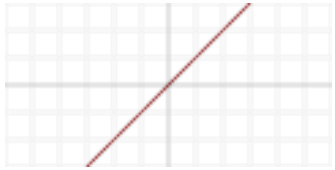
$$O_k = h'\left(\sum_{j=1}^{m} H_j w_{jk} + B_2\right)$$

Deep Learning

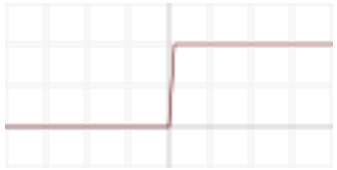
Fundamentos ▶ Elementos

Función de activación h

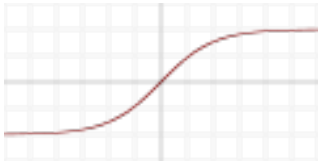
Y. Rebouças (2020) A comprehensive view on activation functions.
https://towardsdatascience.com/a-comprehensive-guide-on-activation-functions-b45ed37a4fa5



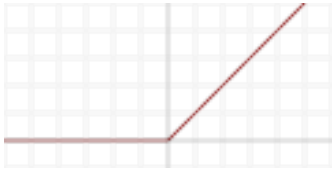
lineal



salto



sinusoidal
/ sigmoide



rectificador lineal (ReLU)

Funciones de salida h'

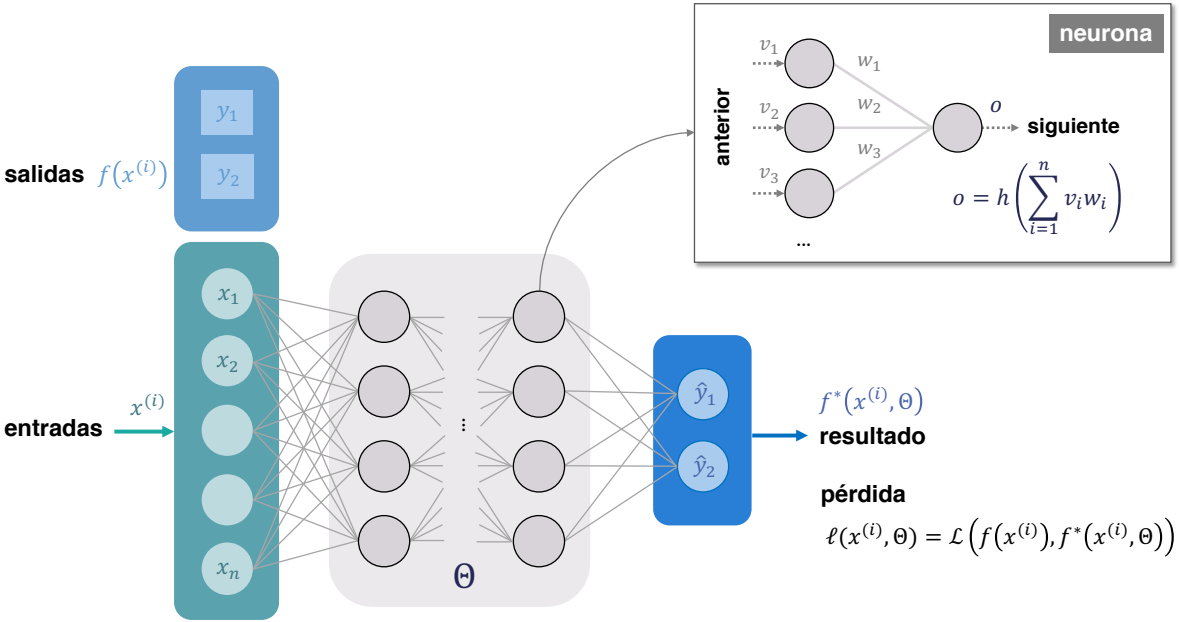
$$f_i(\vec{o}) = \frac{e^{o_i}}{\sum_{j=1}^{|\vec{o}|} e^{o_j}}$$

softmax

$$f(\vec{o}) = \max_i o_i$$

máximo

Error de salida – Función de pérdida



$$MSE = \frac{1}{m}\sum_{j=1}^{m}(y_j - \hat{y}_j)^2$$

error cuadrático medio

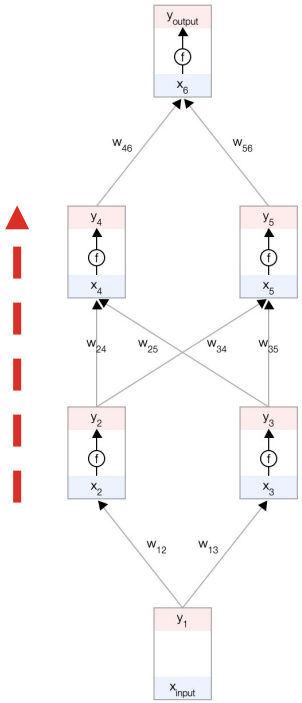
$$H = -\sum_{j=1}^{m} y_j * \log(\hat{y}_j)$$

entropía cruzada

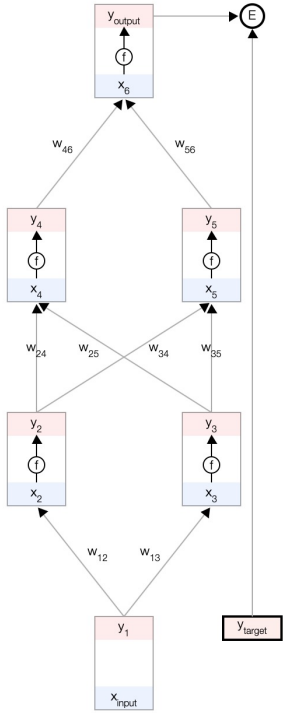
Deep Learning

Fundamentos ► Entrenamiento

Introducción al aprendizaje automático
http://sl.ugr.es/0atf_v2



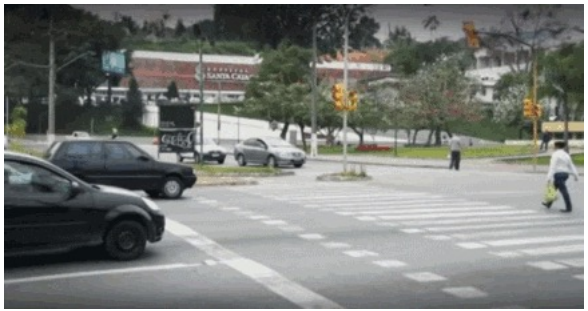
cálculo de la salida (*forward pass*)



$$E(x_{input}, \Theta) = \frac{1}{2}(y_{output} - y_{target})^2$$

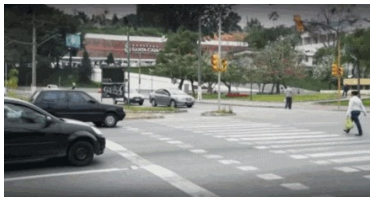
cálculo del error

ajustar Θ para minimizar E



Deep Learning

Fundamentos ► Entrenamiento



datos históricos

	x_1	...	x_n	y_1	...	y_m
$x^{(1)}$						
$x^{(2)}$						
...						
$x^{(K)}$						

random

datos entrenamiento [80%]

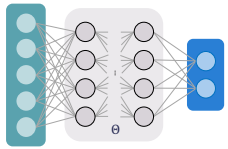
	x_1	...	x_n	y_1	...	y_m
$x^{(i)}$						
...						
$x^{(K')}$						

datos validación [20%]

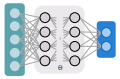
	x_1	...	x_n	y_1	...	y_m
$x^{(i)}$						
$x^{(K'')}$						

$$\Theta = \arg\min_{\Theta} \frac{1}{K'} \sum_{i=1}^{K'} \ell(x^{(i)}, \Theta)$$

optimización



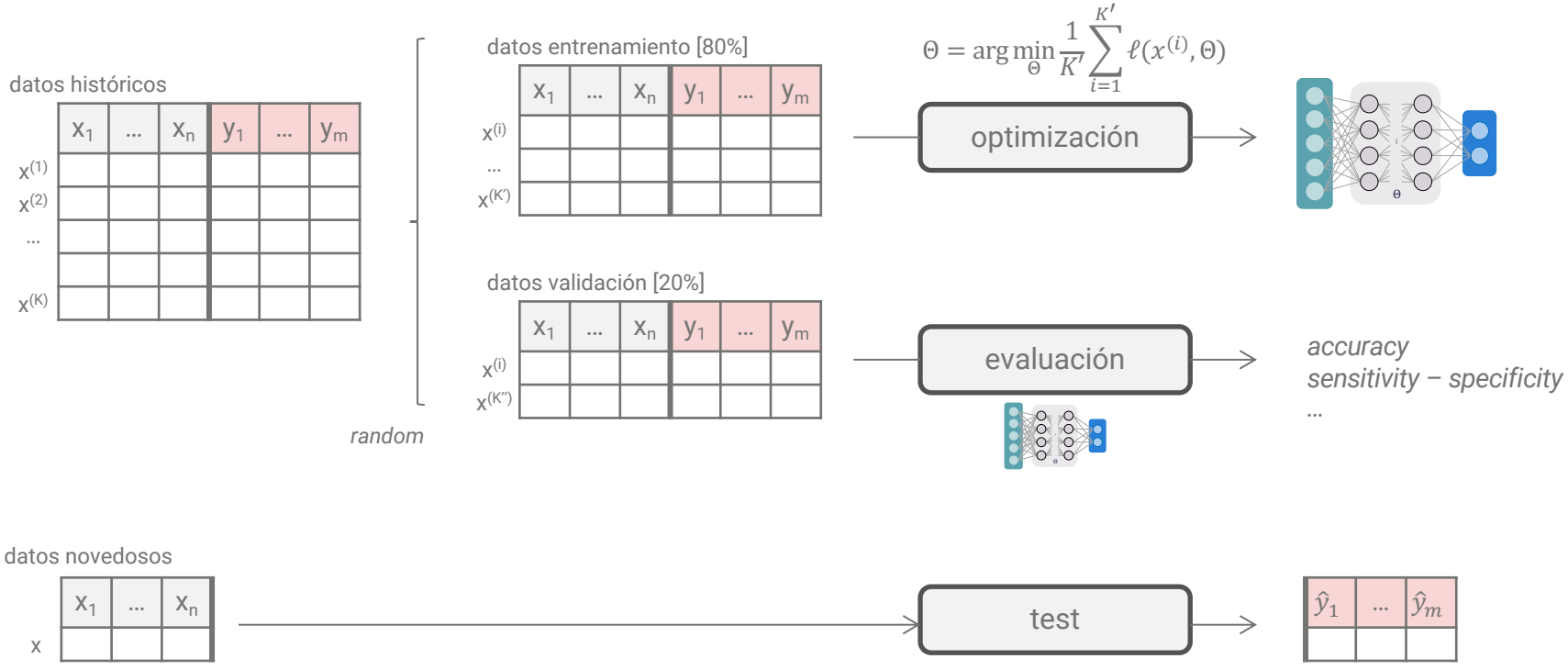
evaluación



accuracy
sensitivity – specificity
...

Deep Learning

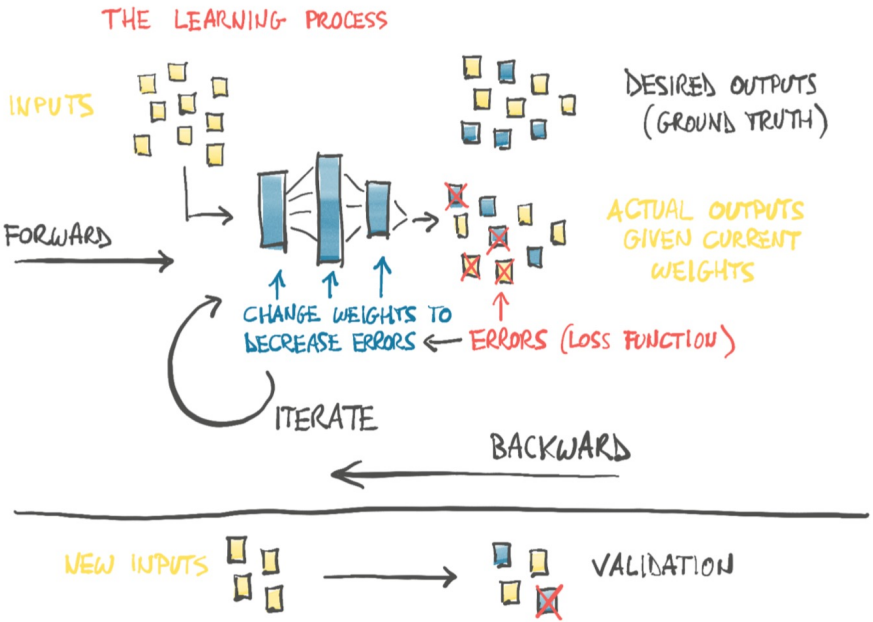
Fundamentos ▶ Entrenamiento



Deep Learning

Fundamentos ▶ Entrenamiento

optimización



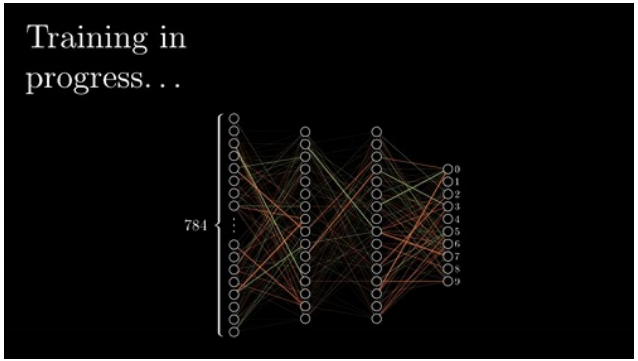
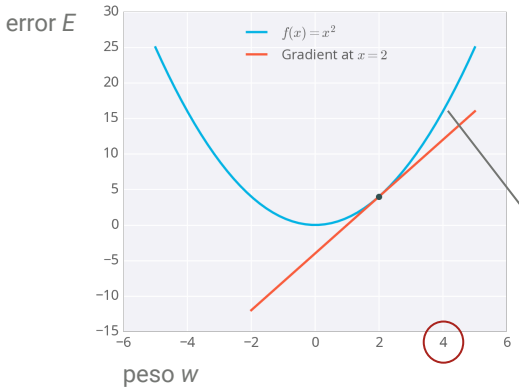


Imagen: 3Blue1Brown

E. Stevens, L. Antiga (2019) Deep Learning with PyTorch. Manning.

Gradiente descendente

La actualización depende del gradiente de la función de error



$$\nabla f = \frac{\partial f(x)}{\partial x}$$

$$f(4) = 16$$
$$\nabla f = 2x$$
$$\nabla f(4) = 8$$

Inicialmente, *w* = 4 , *E*(*w*) = 16
Para buscar el *w* que minimiza *E*
Mover *w* una **cantidad proporcional** a −8

B. Rohrer (2020) How optimization works. https://brohrer.github.io/how_optimization_works_1.html

Deep Learning

Fundamentos ▶ Entrenamiento

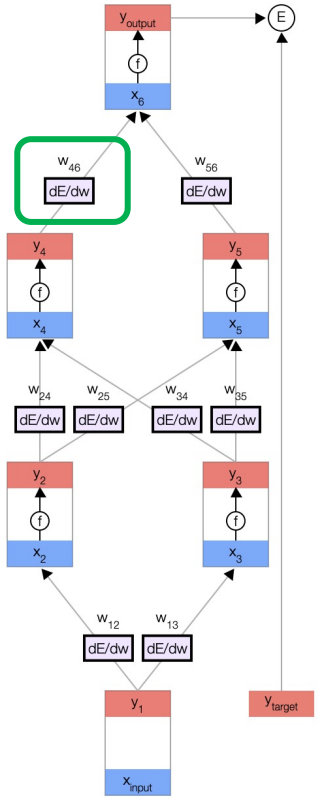
Backpropagation

Actualiza los pesos de la red propagando los errores desde la capa de salida hasta la capa inicial

$$\nabla E = \frac{\partial f(w)}{\partial w}$$

$$\Delta w_{ij} \propto -\frac{\partial E}{\partial w_{ij}}$$

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$



$$E(x_{input}, \Theta) = \frac{1}{2}\left(y_{output} - y_{target}\right)^2$$

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial\left(\frac{1}{2}\left(y - \hat{y}(w)\right)^2\right)}{\partial w_{ij}}$$

$$= -(y - \hat{y})\, h'(x_j)\, y_i$$

Introducción al aprendizaje automático
http://sl.ugr.es/0atf_v2

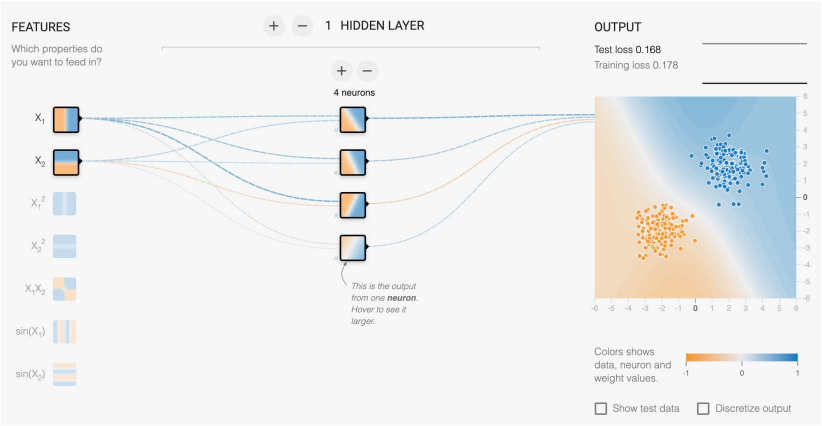
Deep Learning

Fundamentos ▶ Entrenamiento

Carlos Santana Vega (DotCSV)

https://youtu.be/MRIv2IwFTPg

Partes 3, 3.5



Tensorflow Playground

http://sl.ugr.es/tfplayground

Deep Learning

Algoritmos de optimización

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

En cada iteración…

- Gradiente descendente por lotes
  calcula gradientes para todo el conjunto de entrenamiento
- Gradiente descendente estocástico
  calcula gradientes para una sola instancia del conjunto de entrenamiento
- Gradiente descendente (estocástico) con mini-lotes
  calcula gradientes para un subconjunto de instancias del conjunto de entrenamiento

… y actualiza pesos

Tasas de aprendizaje adaptativas:

- Adagrad, Adadelta, RMSProp, Adam, Nadam

S. Ruder (2016) An overview of gradient descent optimization algorithms. https://arxiv.org/abs/1609.04747

Deep Learning

Fundamentos ▶ Desarrollo

Herramientas

Tensorflow [Google]
https://www.tensorflow.org

PyTorch [Facebook]
https://pytorch.org

CNTK [Microsoft]
https://www.microsoft.com/en-us/cognitive-toolkit

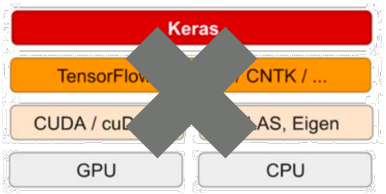
MXNET [Apache]
https://mxnet.incubator.apache.org

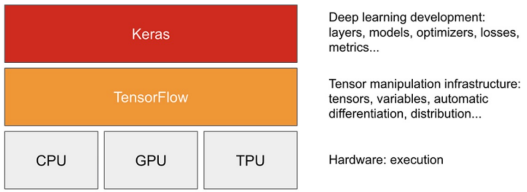
Theano [MILA]
http://deeplearning.net/software/theano/

Keras
https://keras.io

Deep Learning

Fundamentos ▶ Desarrollo





Local

https://tensorflow.rstudio.com/reference/keras/install_keras/

1) CUDA (si GPU)
2) TensorFlow (+ Keras)

```
library(keras)
install_keras(
  method = "conda",            # usar conda
  tensorflow = "default",      # tensorflow = "gpu"
  envname = "r-tensorflow"     # nombre del nuevo entorno
)
```

Amazon EC2

https://tensorflow.rstudio.com/tools/cloud_server_gpu.html

1) Cuenta en AWS
2) Usar AMI RStudio Server with Tensorflow-GPU for AWS, p2.xlarge instance (~ $0.90/hour)
3) Acceder a RStudio Server en el puerto :8787 (rstudio-user / <AWS Instance ID>)

```
library(keras)
```

Google Cloud

Seminario 3 | https://tensorflow.rstudio.com/installation/gpu/cloud_server_gpu/

Docker

https://github.com/rocker-org/ml/

Deep Learning

Fundamentos ▶ Desarrollo

titanic-dl.Rmd

Creacion de la red

```
model <- keras_model_sequential()
model <- model %>%
layer_dense(units = 32, activation = "relu", input_shape = c(ncol(data) - 1)) %>%
layer_dense(units = 16, activation = "relu") %>%
layer_dense(units = 1, activation = "sigmoid")
```

Compilar modelo

```
model %>% compile(
  loss = 'binary_crossentropy',
  metrics = c('accuracy'),
  optimizer = optimizer_adam()
)
```

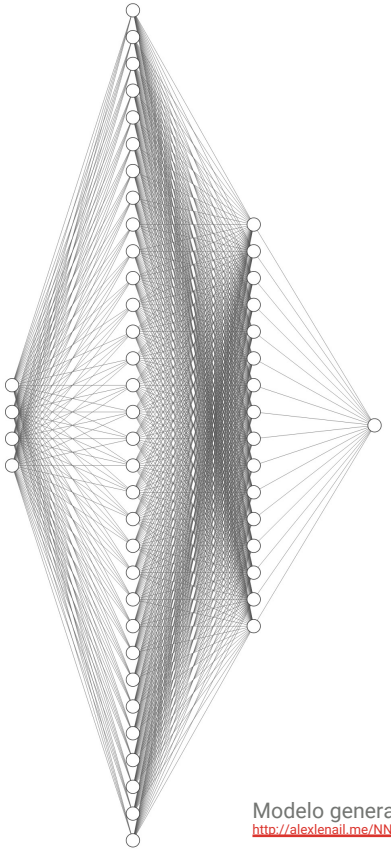
Preparar entrenamiento y validación

Entrenar modelo

```
history <- model %>%
  fit(
  x_train, y_train,
  epochs = 20,
  batch_size = 100,
  validation_split = 0.2
)
```

Deep Learning

Fundamentos ► Desarrollo



Modelo generado con:
http://alexlenail.me/NN-SVG/

Deep Learning

Fundamentos ▶ Desarrollo

Evaluar modelo con conjunto de datos adicional

titanic-dl.Rmd

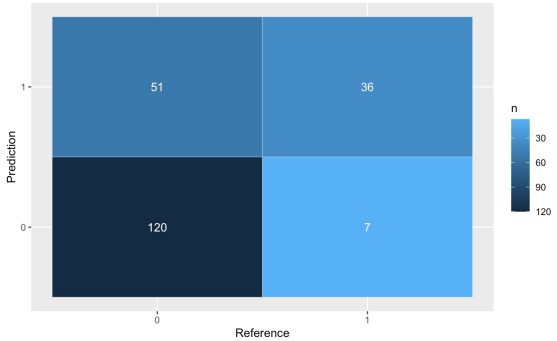
```
model %>% evaluate(x_val, y_val)
```

Predecir con modelo

```
predictions <- model %>% predict_classes(x_val)
```

Validar con conjunto de datos adicional

```
cm <- confusionMatrix(as.factor(y_val), as.factor(predictions))
cm_prop <- prop.table(cm$table)
```



Índice

Deep Learning

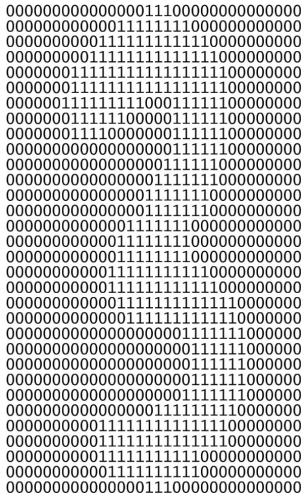
Redes convolutivas

Aproximaciones a la clasificación de imágenes

Codificación directa

MNIST

28



28

784

Multi clasificador

mnist.R

```
$loss
[1] 0.1518216

$acc
[1] 0.9803
```

Y. Lan, L. Lean (2001) Handwritten digit recognition using perceptron neural networks [link]

Deep Learning

Redes convolutivas

Aproximaciones a la clasificación de imágenes

Pre-procesamiento y cálculo de características

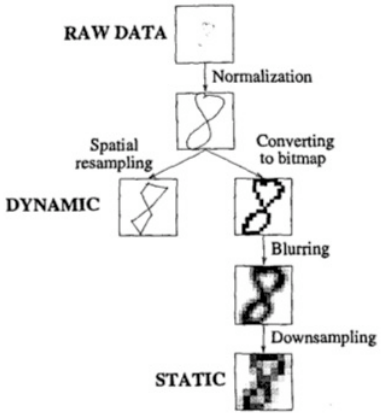
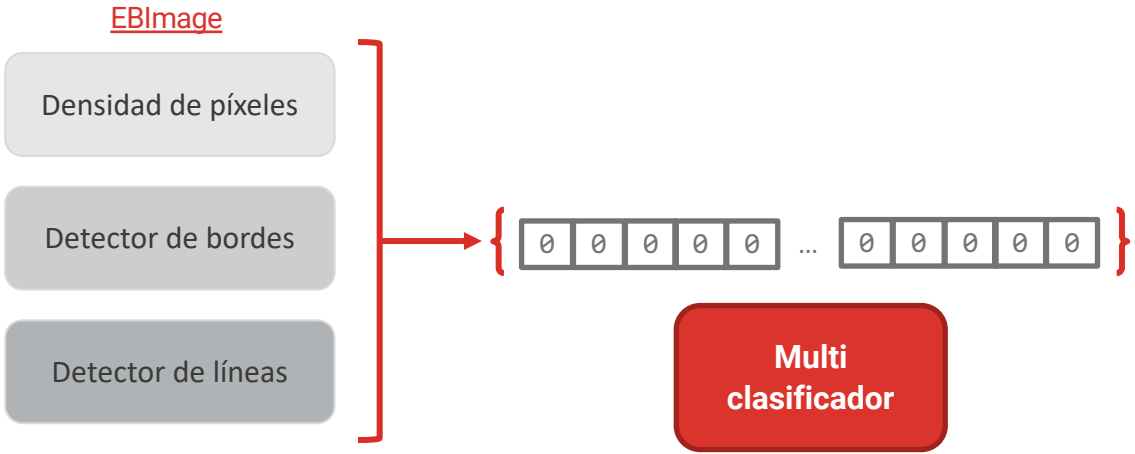


Figure 1: The stages of preprocessing.



F. Alimoglu, E. Alpaydin (1997) Combining multiple representations and classifiers for pen-based handwritten digit recognition. Proceedings of the 4th International Conference on Document Analysis and Recognition.

Deep Learning

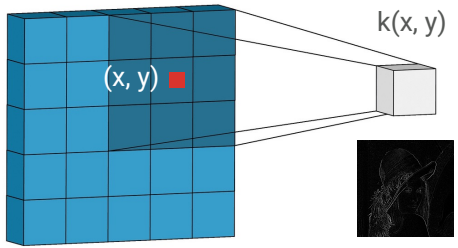
Redes convolutivas

Pre-procesamiento y extracción de características

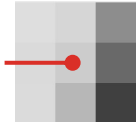
filtro *kernel*

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

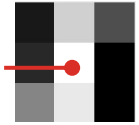


(414, 149)



$$\begin{matrix} 221^{-1} & 217^{-1} & 218^{-1} \\ 213^{-1} & \mathbf{203}^{+8} & 182^{-1} \\ 140^{-1} & 105^{-1} & 66^{-1} \end{matrix}$$

(414, 149)



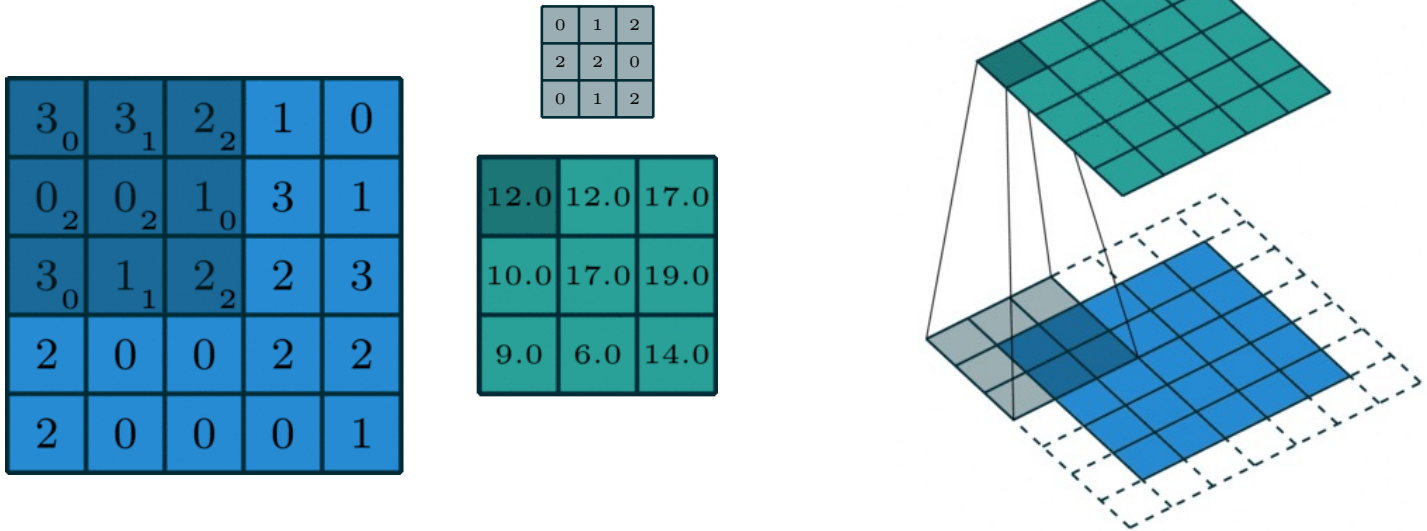
$$\begin{matrix} 30 & 44 & 132 \\ 207 & \mathbf{255} & 232 \\ 79 & 0 & 0 \end{matrix}$$

k(414, 149) = 8x203−221−217−218−213−182−140−105−66

Deep Learning

Redes convolutivas

Pre-procesamiento y extracción de características



I. Shafkat (2018) Intuitively Understanding Convolutions for Deep Learning [link]

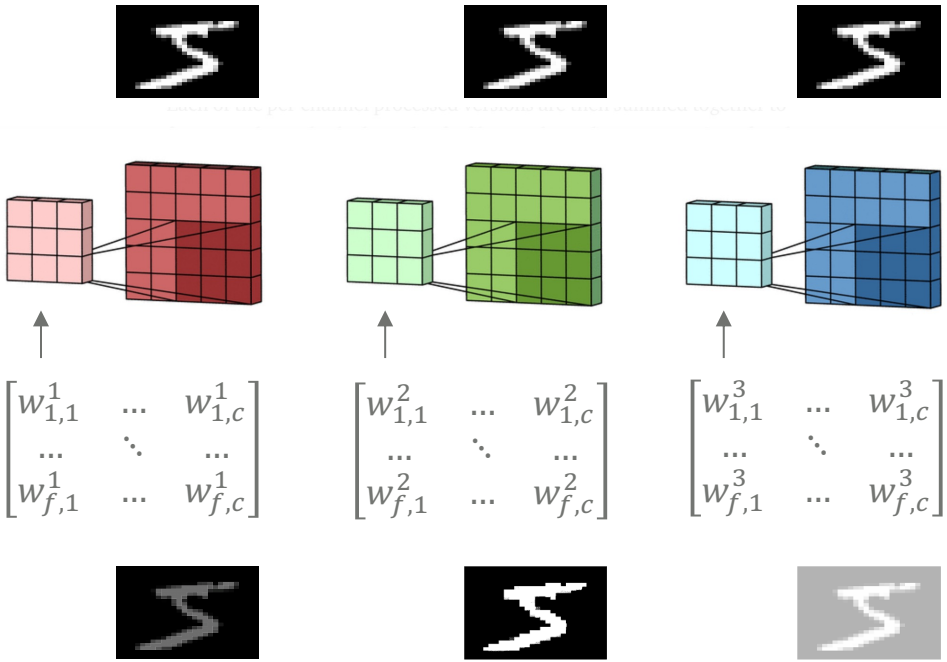
V. Dumoulin, F. Visin (2018) A guide to convolution arithmetic for deep learning [link]

Deep Learning

Redes convolutivas

Red convolutiva

Capas de convolución



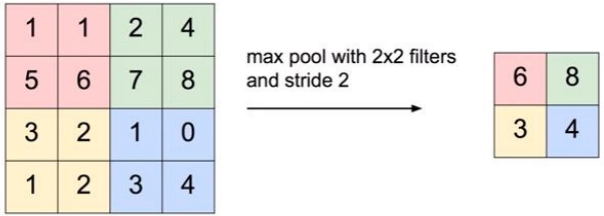
Deep Learning

Redes convolutivas

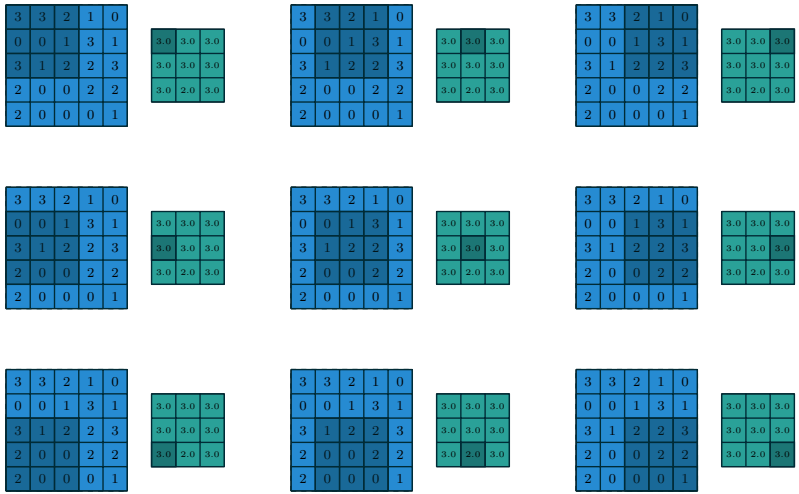
Red convolutiva

Pooling

max-pooling 2x2, stride = 2



max-pooling 3x3, stride = 1

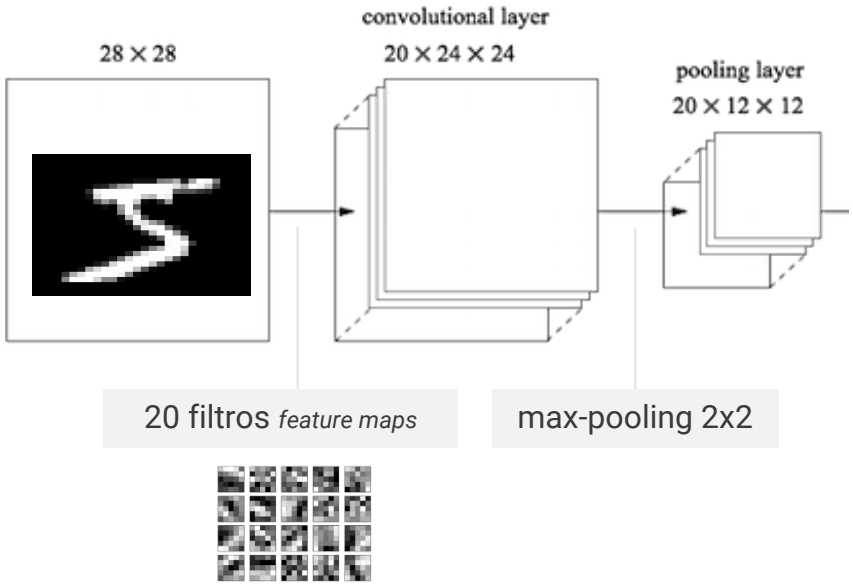


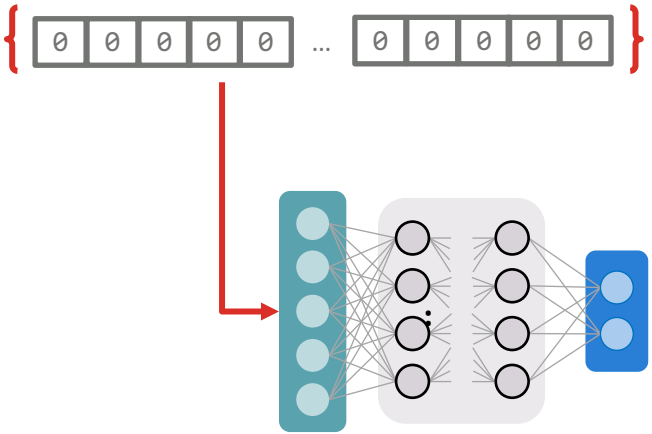
Deep Learning

Redes convolutivas

Red convolutiva

Fully-connected (densa)



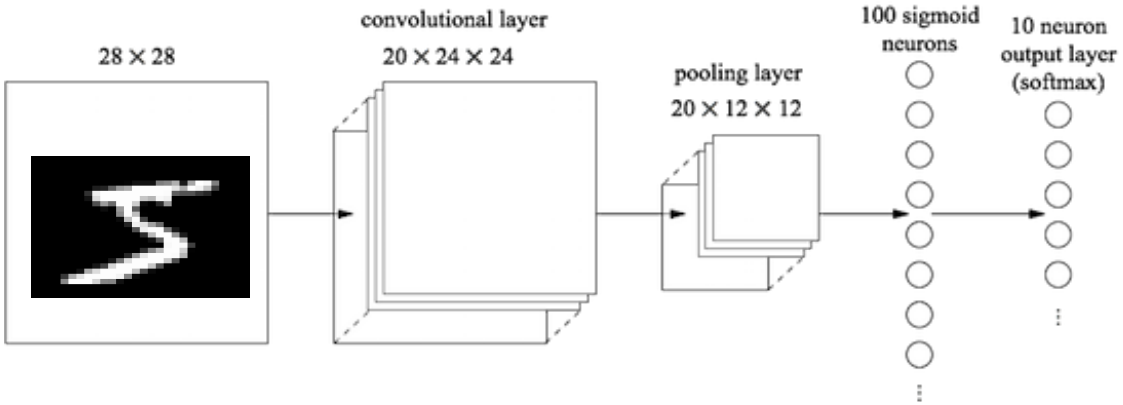


Deep Learning

Redes convolutivas

Red convolutiva

Fully-connected



mnist-cnn.R

```
model <- keras_model_sequential()
model %>%
  layer_conv_2d(filters = 20, kernel_size = c(5, 5), activation = "relu", input_shape = c(28, 28, 1)) %>%
  layer_max_pooling_2d(pool_size = c(2, 2)) %>%
  layer_flatten() %>%
  layer_dense(units = 100, activation = "sigmoid") %>%
  layer_dense(units = 10, activation = "softmax")
```

```
$loss
[1] 0.04407667

$acc
[1] 0.9854
```

Deep Learning

Redes convolutivas

Arquitectura

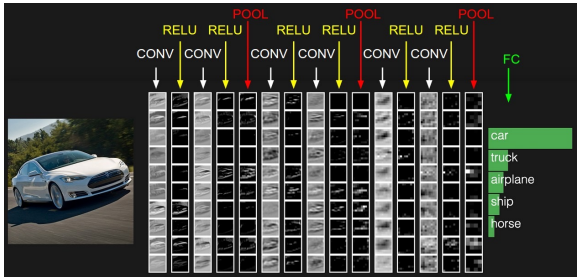
INPUT (entrada)

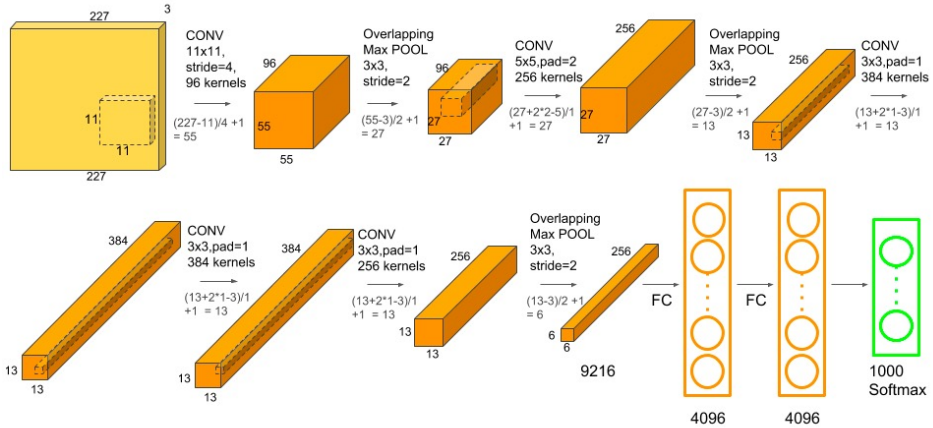
CONV (convolución)

RELU (activación)

POOL (reducción de dimensionalidad)

FC (totalmente conectada)





AlexNet (2012)

15.3% error en ImageNet

(siguiente: 26.2%)

https://www.learnopencv.com/understanding-alexnet/

Índice

Deep Learning

Mejora del aprendizaje

Nota sobre reproducibilidad de resultados en R Keras

El proceso de entrenamiento de una red neuronal en Keras comprende varios pasos que introducen aleatoriedad en el proceso y dificultan la reproducibilidad de los resultados

- Inicialización aleatoria de los pesos de la red
- Uso de gradiente descendente estocástico con mini-lotes
- Otros métodos con aleatoriedad (por ejemplo, *dropout)*
- Ejecución paralela en GPU
- etc.

Es posible conseguir resultados más reproducibles fijando algunas de las semillas aleatorias que utiliza Keras con el *backend* de Tensorflow `use_session_with_seed`

Esta función **desactiva el uso de GPU**

Los ejemplos que se muestran en el resto del tema **NO** aseguran reproducibilidad. Por lo tanto, en ocasiones puedes obtener salidas diferentes con el mismo fichero

En una experimentación exhaustiva, se proporcionarían valores medios de las métricas obtenidos con múltiples repeticiones

Deep Learning

Mejora del aprendizaje

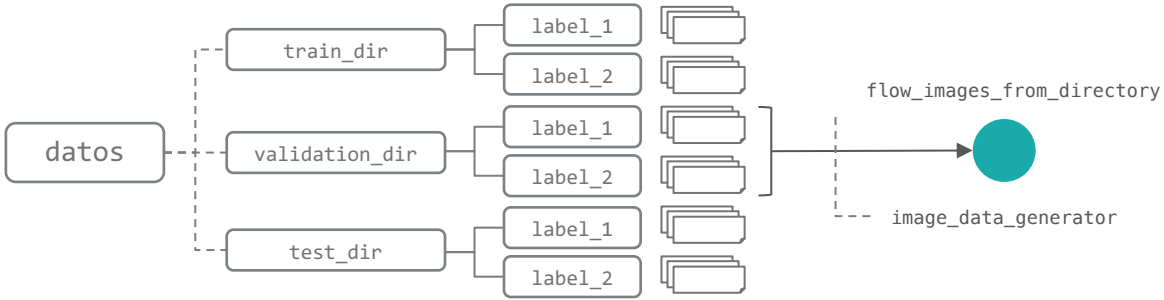
Perros y gatos

Carga de datos

dogsVScats.R

```
# https://tensorflow.rstudio.com/keras/reference/image_data_generator.html
train_datagen         <- image_data_generator(rescale = 1/255)
validation_datagen    <- image_data_generator(rescale = 1/255)
test_datagen          <- image_data_generator(rescale = 1/255)

# https://tensorflow.rstudio.com/keras/reference/flow_images_from_directory.html
train_data <- flow_images_from_directory(
        directory = train_dir,
        generator = train_datagen,
        target_size = c(150, 150),    # (w, h) --> (150, 150)
        batch_size = 20,              # grupos de 20 imágenes
        class_mode = "binary"         # etiquetas binarias
)
```

IMAGENET



Deep Learning

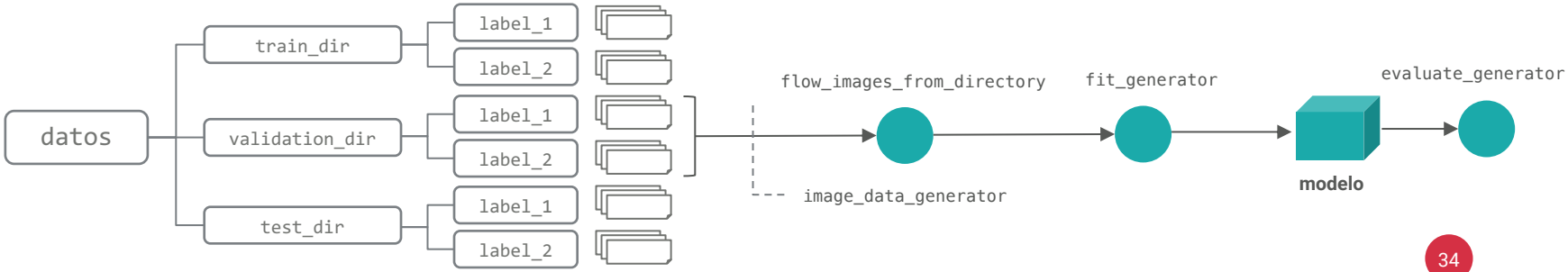
Mejora del aprendizaje

Perros y gatos

Entrenamiento, validación y test

dogsVScats.R

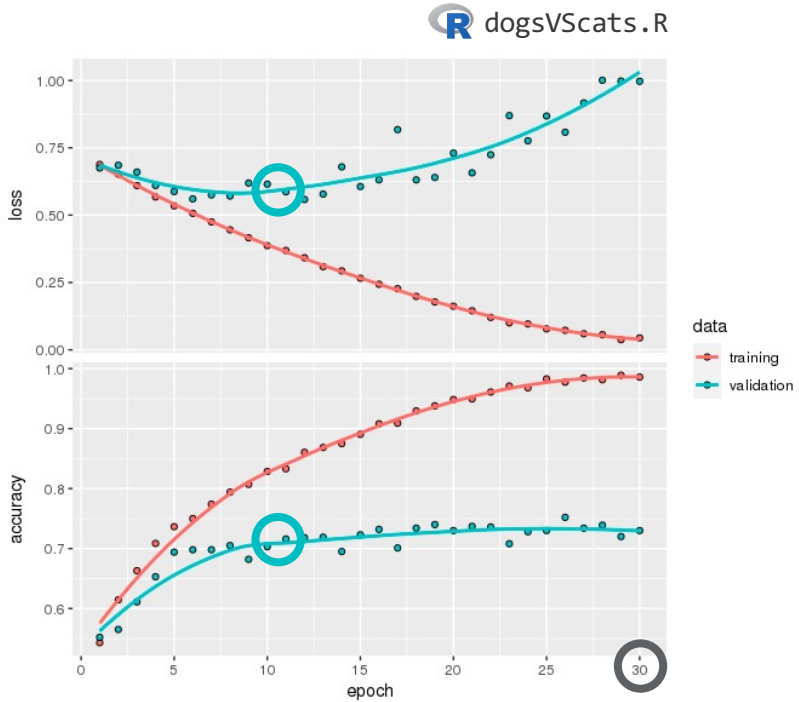
```
# https://tensorflow.rstudio.com/keras/reference/fit_generator.html
history <- model %>%
        fit_generator(
                train_data,
                steps_per_epoch = 100,
                epochs = 30,
                validation_data = validation_data,
                validation_steps = 50
        )

# https://tensorflow.rstudio.com/keras/reference/evaluate_generator.html
model %>% evaluate_generator(test_data, steps = 50)
```



Deep Learning

Mejora del aprendizaje

Perros y gatos




```
Train
$loss
[1] 0.0443

$accuracy
[1] 0.9860

-----
Validation
$loss
[1] 0.9970

$accuracy
[1] 0.7300

-----
Test
$loss
[1] 0.960888

$accuracy
[1] 0.73
```

Deep Learning

Mejora del aprendizaje

Sobreaprendizaje *(overfitting)*

Un modelo es capaz de predecir adecuadamente las salidas para los valores de entrenamiento, pero no para los de validación (y, probablemente, tampoco para los de test)

El modelo no tiene capacidad de generalización

Imagina que tenemos un robot al que queremos enseñar a reconocer fotos de gatos. Para ello, le mostramos 100 fotos de gatos y 100 fotos de perros (*entrenamiento*).

Una vez estudiadas las fotos, el robot dice que ya es capaz de identificar gatos. Si le mostramos una foto de las que ya conoce, seguramente la clasificará correctamente. Sin embargo, si queremos saber si el robot es bueno en su trabajo, deberíamos mostrarle otras fotos que no ha visto *(validación / test*).

¡El robot podría haber guardado en su memoria las fotos que le hemos pasado y simplemente limitarse a buscar si tiene memorizada una **IGUAL**!

https://youtu.be/7-6X3DTt3R8
DotCSV – Partes 1, 2

Deep Learning

Perros y gatos

Aumento de datos

dogsVScats_augmentation.R

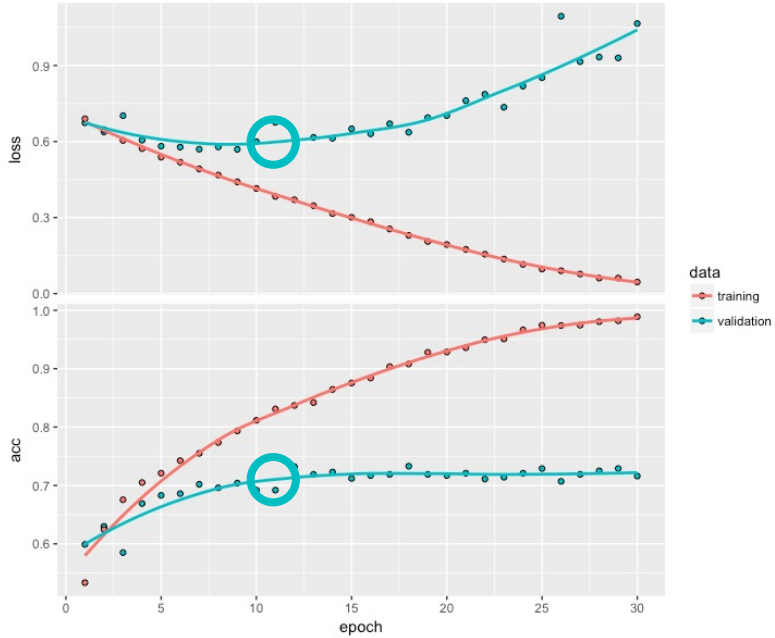
```
# https://tensorflow.rstudio.com/keras/reference/image_data_generator.html
data_augmentation_datagen <- image_data_generator(
        rescale = 1/255,
        rotation_range = 40,
        width_shift_range = 0.2,
        height_shift_range = 0.2,
        shear_range = 0.2,
        zoom_range = 0.2,
        horizontal_flip = TRUE,
        fill_mode = "nearest"
)

# https://tensorflow.rstudio.com/keras/reference/fit_generator.html
history <- model %>%
        fit_generator(
                train_augmented_data,
                steps_per_epoch = 100,
                epochs = 30,
                validation_data = validation_data,
                validation_steps = 50
        )
```

Deep Learning

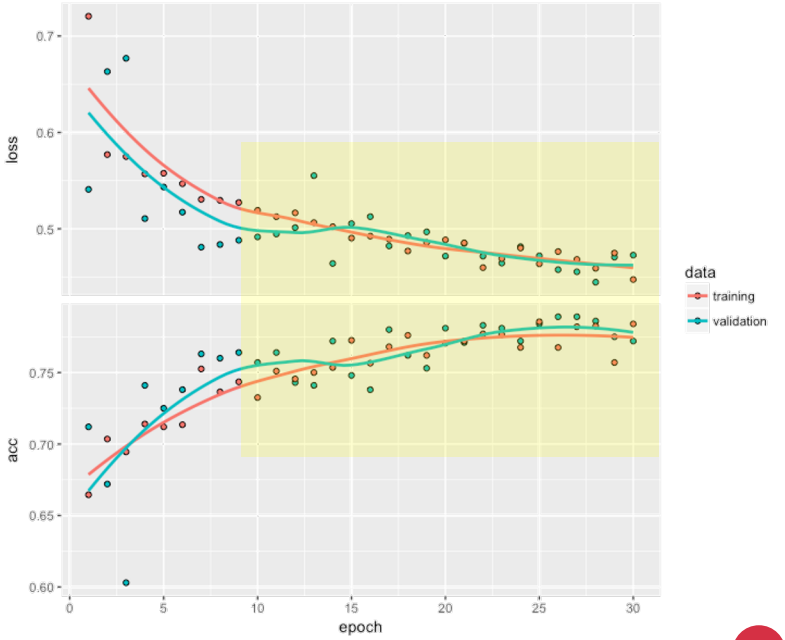
Mejora del aprendizaje ► Aumento de datos

Perros y gatos

dogsVScats.R



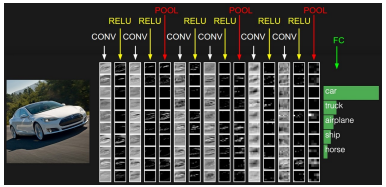
dogsVScats_augmentation.R



Deep Learning

Mejora del aprendizaje

Sobreaprendizaje *(overfitting)*



Selección de datos de entrenamiento y aumento de datos

Incrementar el número de datos disponibles y asegurar que son representativos

Early-stopping

Detener el entrenamiento cuando el error de validación comienza a aumentar

Regularización

Modificar la forma en que se actualizan los pesos de la red durante el entrenamiento para que mantengan valores dentro de un rango

Normalización

Modifican la salida de una neurona para que se mantenga dentro de un rango

Dropout

Poner a 0 pesos de la red para aumentar resistencia al aprendizaje

Early stopping

Detener el entrenamiento cuando el error de validación comienza a aumentar

Dificultades

No hay un criterio universalmente definido

Qué significa que el error de validación comienza a aumentar

Puede aumentar el tiempo dedicado a entrenar el modelo

Más pruebas, más supervisión

Riesgo de caída en óptimos locales

No se explora la región del espacio donde está el óptimo

Puede ser contraproducente

Interesa que los parámetros de la red se estabilicen en una región amplia

Deep Learning

Early stopping

Detener el entrenamiento cuando el error de validación comienza a aumentar

Soporte en Keras

- Grabación de modelos
  `save_model_hdf5`, `save_model_tf`, `load_model_hdf5`, `load_model_tf`
  `save_model_weights_hdf5`, `load_model_weights_hdf5`
- Continuación de entrenamiento a partir de los pesos actuales del modelo (*checkpoints*)
- *Callbacks* en función `fit`

  Guardar modelo conforme se va entrenando
  `callback_model_checkpoint` [ejemplo]

  Detener entrenamiento
  `callback_early_stopping`

dogsVScats_early-stopping.R

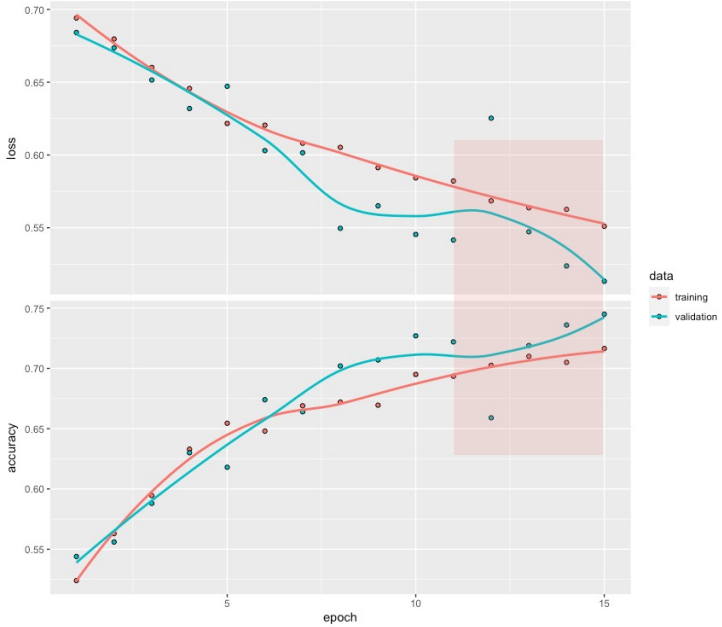
```
# https://keras.rstudio.com/reference/callback_early_stopping.html
history <- model %>%
       fit_generator(
              train_augmented_data,
              steps_per_epoch = 100,
              epochs = 30,
              validation_data = validation_data,
              validation_steps = 50,
              callbacks = list(
                     callback_early_stopping(patience = 2)))
```

Deep Learning

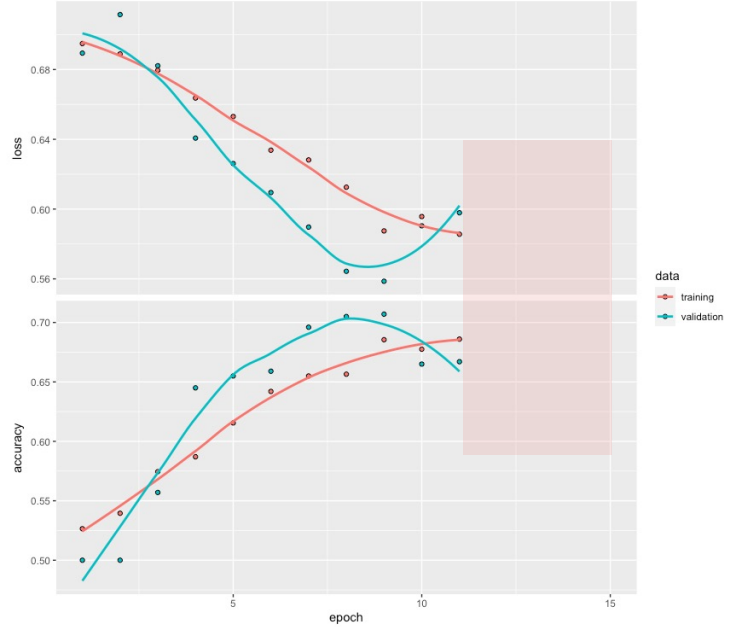
Mejora del aprendizaje ► *Early stopping*

Perros y gatos

dogsVScats_augmentation.R



dogsVScats_early-stopping.R



Deep Learning

Regularización

Modificar la forma en que se actualizan los pesos de la red durante el entrenamiento para que mantengan valores en un determinado rango

Regularización *weight decay*

Durante el entrenamiento, se añade un 'coste adicional' al coste si los pesos son altos

L1 regularization *lasso*
El coste añadido es proporcional al valor absoluto de los pesos

$$\Omega(\theta) = \lambda \sum |w|$$

L2 regularization *ridge*
El coste añadido es proporcional al cuadrado de los valores de los pesos

$$\Omega(\theta) = \lambda \sum w^2$$

Dónde: Capas densas, se prefiere L2 a L1 se puede usar también en convolutivas, pero su efectividad es discutible

Soporte en Keras

Parámetro `kernel_regularizer` al definir una capa
Posibilidad de definir regularizadores a medida

TensorFlow
Lattice [link]

Deep Learning

Normalización

Modifican la salida de una neurona para que se mantenga dentro de un rango

Consiguen modelos menos dependientes del preprocesamiento y de la inicialización de los pesos

Normalización por lotes (*batch normalization*)

Se normalizan las activaciones de una neurona obtenidas con un mini-lote de tamaño k antes de pasarlas a la siguiente capa:

$$H'_j = \frac{H_j - \mu}{\sigma}$$

$$\mu = \frac{1}{k}\sum_i H_i \qquad \sigma = \sqrt{\varepsilon + \frac{1}{k}\sum_i (H_i - \mu)^2}$$

Dónde: A continuación de capas densas y capas convolutivas, normalmente después de la activación también es posible antes de la función de activación, sujeto a discusión

Soporte en Keras

Capa `layer_batch_normalization`

Deep Learning

Mejora del aprendizaje ► *Dropout*

Dropout

Poner a 0 pesos de la red durante el entrenamiento para aumentar resistencia al aprendizaje

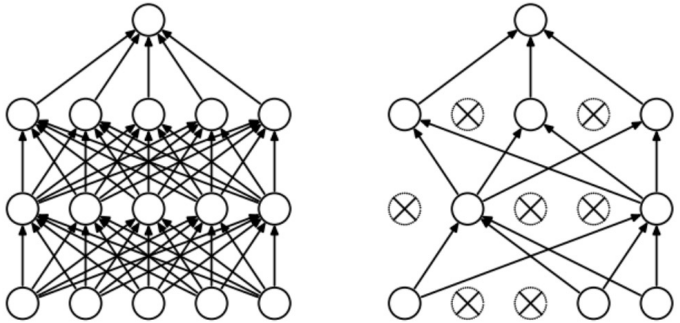
Se seleccionan aleatoriamente con probabilidad *p*

Dónde: Se prefiere en capas densas (*p*=0.5). Puede usarse también en las capas convolutivas (*p*=0.1)

No se utiliza junto a regularización

Soporte en Keras

- Capa `layer_dropout`



(a) Standard Neural Net (b) After applying dropout.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov (2014) Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research 15(56): 1929–1958 [link]

S. Park, N. Kwak (2016) Analysis on the dropout effect in convolutional neural networks. Asian Conference on Computer Vision (ACCV 2016), pp. 189–204 [link]

Deep Learning

Mejora del aprendizaje

MNIST-CNN

mnist-cnn.R

```
model %>%
        layer_conv_2d(filters = 20, kernel_size = c(5, 5), activation = "relu", input_shape = c(28, 28, 1)) %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_flatten() %>%
        layer_dense(units = 100, activation = "sigmoid") %>%
        layer_dense(units = 10, activation = "softmax")
```

```
Test
$loss                  $accuracy
[1] 0.04481287         [1] 0.986
```

mnist-cnn_mejorado.R

```
model %>%
        layer_conv_2d(filters = 20, kernel_size = c(5, 5), activation = "relu", input_shape = c(28, 28, 1)) %>%
        layer_batch_normalization(epsilon = 0.01) %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_flatten() %>%
        layer_dense(units = 100, activation = "sigmoid", kernel_regularizer = regularizer_l2(0.01)) %>%
        # layer_dropout(rate = 0.4) %>%
        layer_dense(units = 10, activation = "softmax")
```

```
Test
$loss                  $accuracy
[1] 0.1622155          [1] 0.9718
```

Deep Learning

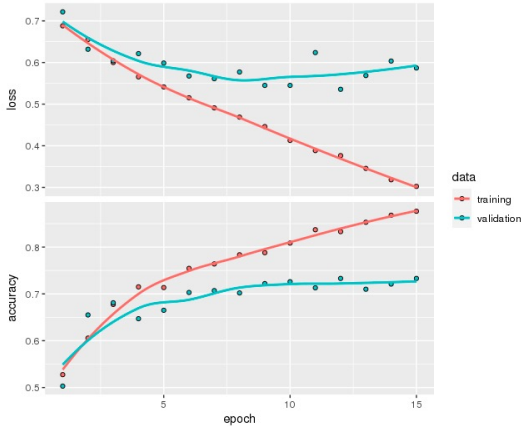
Mejora del aprendizaje

dogs VS cats

dogsVScats.R

```
model <- keras_model_sequential() %>%
        layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu", input_shape = c(150, 150, 3)) %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_flatten() %>%
        layer_dense(units = 512, activation = "relu") %>%
        layer_dense(units = 1, activation = "sigmoid")
```

```
Test
$loss
[1] 0.5747627

$accuracy
[1] 0.732
```



Deep Learning

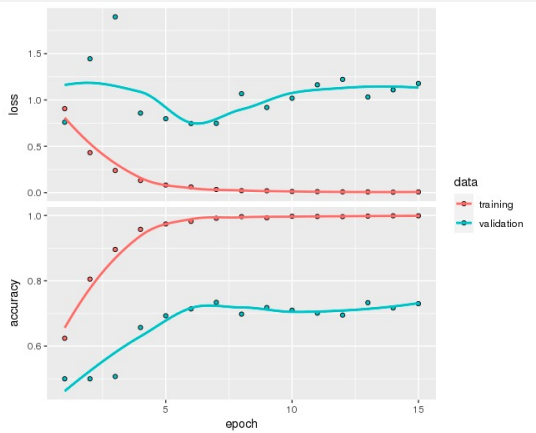
Mejora del aprendizaje

dogs VS cats

dogsVScats_mejorado.R

```
model <- keras_model_sequential() %>%
        layer_conv_2d(filters = 32, kernel_size = c(3, 3), activation = "relu", input_shape = c(150, 150, 3)) %>%
        layer_batch_normalization(epsilon = 0.01) %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_conv_2d(filters = 64, kernel_size = c(3, 3), activation = "relu") %>%
        layer_batch_normalization(epsilon = 0.01) %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
        layer_batch_normalization(epsilon = 0.01) %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_conv_2d(filters = 128, kernel_size = c(3, 3), activation = "relu") %>%
        layer_batch_normalization(epsilon = 0.01) %>%
        layer_max_pooling_2d(pool_size = c(2, 2)) %>%
        layer_flatten() %>%
        layer_dense(units = 512, activation = "relu") %>%
        layer_dropout(rate = 0.3) %>%
        layer_dense(units = 1, activation = "sigmoid")
```

```
Test
$loss
[1] 1.21541

$accuracy
[1] 0.716
```



Índice

Deep Learning

Gestión del proceso de aprendizaje

tfruns

https://tensorflow.rstudio.com/tools/tfruns/overview/

Tensorboard
https://youtu.be/2vh_Jk2wALU

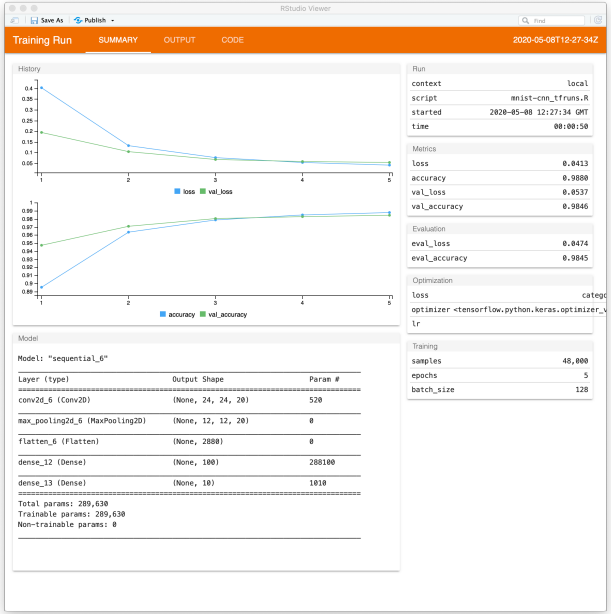
Seguimiento y visualización del entrenamiento

tfruns.R

```
# Lanzar script de entrenamiento
training_run("mnist-cnn_tfruns.R")

# Consultar resultados (modo texto)
latest_run()

# Visualizar ejecución en navegador
view_run()
```

RStudio > Add-ins

```
$ RScript - 'tfruns::training_run("mnist-cnn_tfruns.R")
```



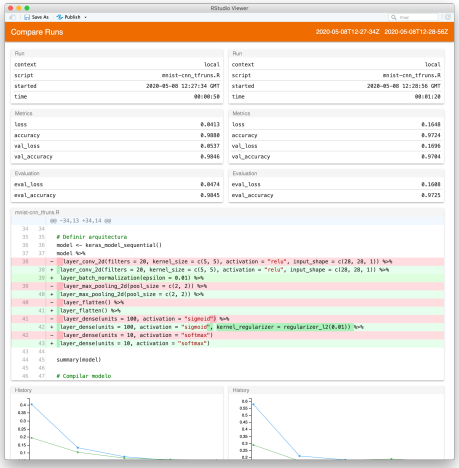
Deep Learning

Gestión del proceso de aprendizaje

tfruns — Comparar ejecuciones

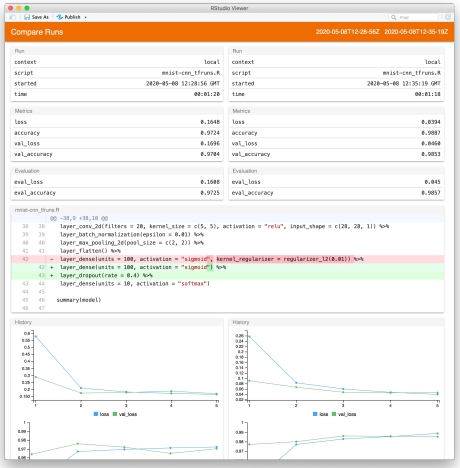
tfruns.R

```
# Comparar las dos últimas ejecuciones
compare_runs()
```



02 compare-runs-2020-05-08T12-28-56Z-2020-05-08T12-27-34Z.html

inicial **vs** L2



03 compare-runs-2020-05-08T12-35-19Z-2020-05-08T12-28-56Z.html

L2 **vs** *dropout*

Deep Learning

Gestión del proceso de aprendizaje

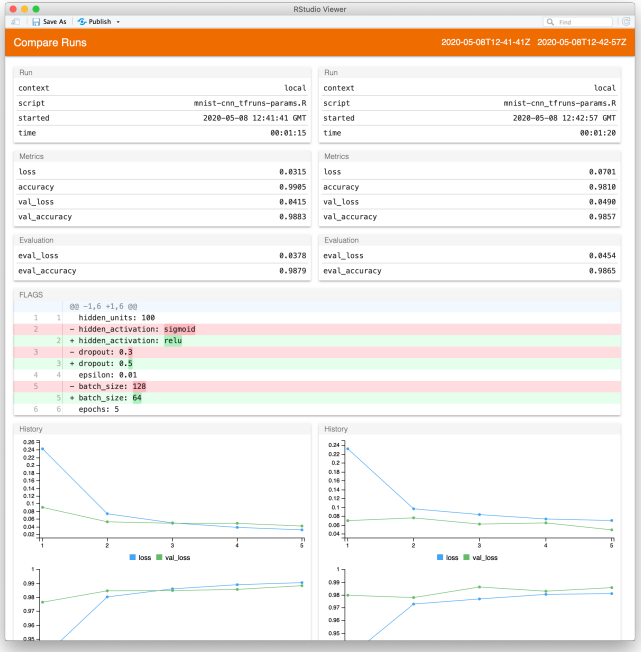
tfruns — Parametrizar ejecuciones

tfruns.R

```
# Lanzar script parametrizado
training_run("mnist-cnn_tfruns-params.R")

# Lanzar script parametrizado con nuevos parámetros
training_run("mnist-cnn_tfruns-params.R",
             flags = list(
                  dropout=0.5,
                  hidden_activation ="relu",
                  batch_size=64))
```

mnist-cnn_tfruns-params.R

```
FLAGS <- flags(
          flag_numeric("hidden_units", 100),
          flag_string("hidden_activation", "sigmoid"),
          flag_numeric("dropout", 0.3),
          flag_numeric("epsilon", 0.01),
          flag_numeric("batch_size", 128),
          flag_numeric("epochs", 5)
)

# ...

model <- keras_model_sequential()
model %>%
          layer_conv_2d(filters = 20, kernel_size = c(5, 5),
                        activation = "relu", input_shape = c(28, 28, 1)) %>%
          layer_batch_normalization(epsilon = FLAGS$epsilon) %>%
          layer_max_pooling_2d(pool_size = c(2, 2)) %>%
          layer_flatten() %>%
          layer_dense(units = FLAGS$hidden_units, activation = FLAGS$activation) %>%
          layer_dropout(rate = FLAGS$dropout) %>%
          layer_dense(units = 10, activation = "softmax")
```



04 compare-runs-2020-05-08T12-42-57Z-2020-05-08T12-41-41Z.html

por defecto **vs** *dropout=0.5 + relu + batch=64*

Deep Learning

Gestión del proceso de aprendizaje

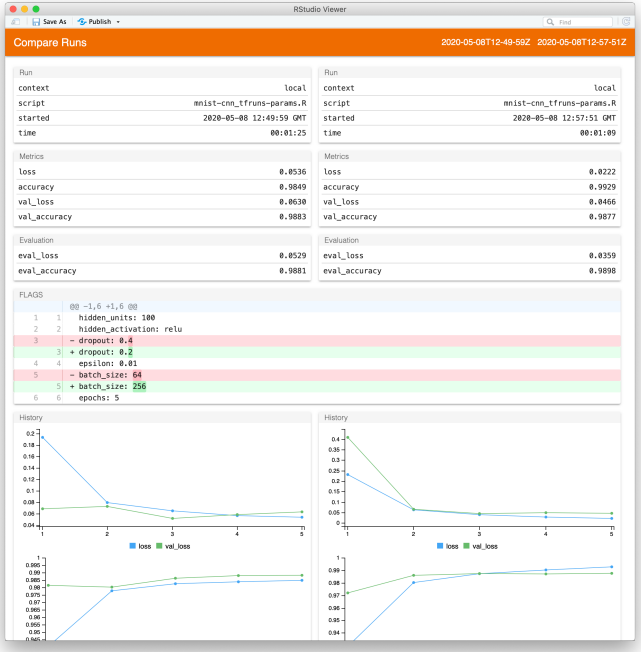
tfruns Lanzar conjuntos de experimentos parametrizados

tfruns.R

```
# Lanzar conjunto de experimentos parametrizados
runs <- tuning_run("mnist-cnn_tfruns-params.R",
          runs_dir = "dropout_batch_tuning",
          flags = list(dropout=c(0.2, 0.3, 0.4, 0.5),
          hidden_activation ="relu",
          batch_size=c(64, 128, 256))) # solo una muestra: sample = 0.3

# Listar por orden de resultado
runs[order(runs$eval_accuracy, decreasing = TRUE), ]
```

mnist-cnn_tfruns-params.R

```
FLAGS <- flags(
          flag_numeric("hidden_units", 100),
          flag_string("hidden_activation", "sigmoid"),
          flag_numeric("dropout", 0.3),
          flag_numeric("epsilon", 0.01),
          flag_numeric("batch_size", 128),
          flag_numeric("epochs", 5)
)

# ...

model <- keras_model_sequential()
model %>%
          layer_conv_2d(filters = 20, kernel_size = c(5, 5),
                        activation = "relu", input_shape = c(28, 28, 1)) %>%
          layer_batch_normalization(epsilon = FLAGS$epsilon) %>%
          layer_max_pooling_2d(pool_size = c(2, 2)) %>%
          layer_flatten() %>%
          layer_dense(units = FLAGS$hidden_units, activation = FLAGS$activation) %>%
          layer_dropout(rate = FLAGS$dropout) %>%
          layer_dense(units = 10, activation = "softmax")
```



05 compare-runs-2020-05-08T12-57-51Z-2020-05-08T12-49-59Z.html

dropout=0.4 *batch*=64 **vs** *dropout*=0.2 *batch*=256

Deep Learning

Gestión del proceso de aprendizaje

mlflow

https://mlflow.org/docs/latest/quickstart.html

Soporta múltiples lenguajes de programación (R) y bibliotecas de Aprendizaje Automático

Mayor potencia y flexibilidad, pero también más complejidad de uso

Ofrece integraciones pre-implementadas (`autolog`): Keras, scikit-learn, h20, PyTorch, SpaCy, etc.

Permite realizar despliegue de aplicaciones

Elementos

Tracking: registro de experimentos (código, datos, configuración, resultados, etc.)

Projects: empaquetado de código para reproducir ejecuciones en cualquier plataforma

Models: despliegue de modelos en diversas plataformas (local y cloud)

Registry: repositorio de modelos entrenados

Deep Learning

Gestión del proceso de aprendizaje

mlflow Lanzar conjuntos de experimentos parametrizados

mlflow.R

```r
# Lanzar script de entrenamiento
mlflow_run(entry_point = "mnist-cnn_mlflow.R")

# Visualizar en interfaz MLflow
# http://127.0.0.1:5987/
mlflow_ui()

# Cambiar valores de los parámetros
mlflow_run(entry_point = "mnist-cnn_mlflow.R", parameters =
list(dropout = 0.5, epochs = 3))
```

mnist-cnn_mlflow.R

```r
hidden_units <- mlflow_param(
   "hidden_units", 100, "integer", "Number of units of the hidden layer")
hidden_activation <- mlflow_param(
   "hidden_activation", "sigmoid", "string", "Activation function for the
    hidden layer")
dropout <- mlflow_param(
   "dropout", 0.3, "numeric", "Dropout rate (after the hidden layer)")
epsilon <- mlflow_param(
   "epsilon", 0.01, "numeric", "Epsilon parameter of the batch
    normalization (after convolution)")
batch_size <- mlflow_param(
   "batch_size", 128, "integer", "Mini-batch size")
epochs <- mlflow_param(
   "epochs", 5, "integer", "Number of training epochs")
```

mnist-cnn_mlflow.R

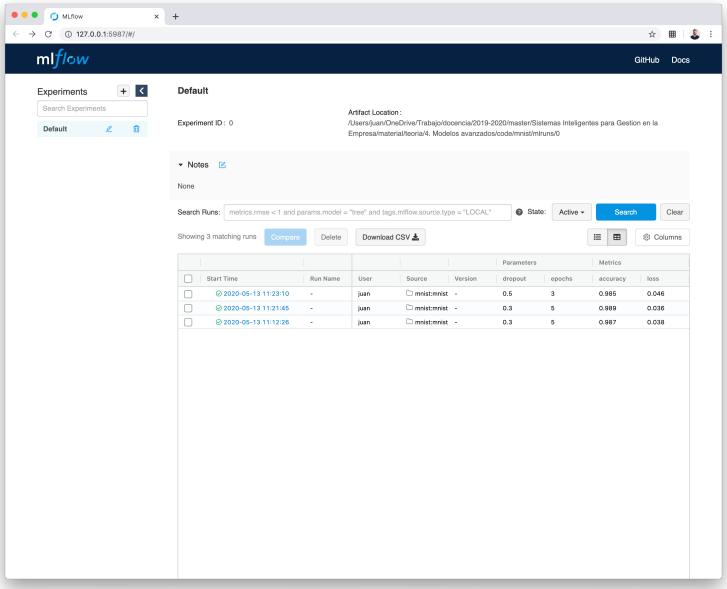
```r
with(mlflow_start_run(), {

   # Entrenar modelo
   history <- model %>%
      fit(
         x_train, y_train,
         epochs = epochs,
         batch_size = batch_size,
         validation_split = 0.2
   )

   # Calcular metricas sobre datos de validación
   metrics <- model %>%
      evaluate(x_test, y_test)

   # Guardar valores interesantes de la ejecución
   # Por ejemplo, para estudio de dropout + epochs
   # parámetros
   mlflow_log_param("dropout", dropout)
   mlflow_log_param("epochs", epochs)

   # métricas
   mlflow_log_metric("loss", metrics$loss)
   mlflow_log_metric("accuracy", metrics$accuracy)

   # modelo
   mlflow_log_model(model, "model")
```

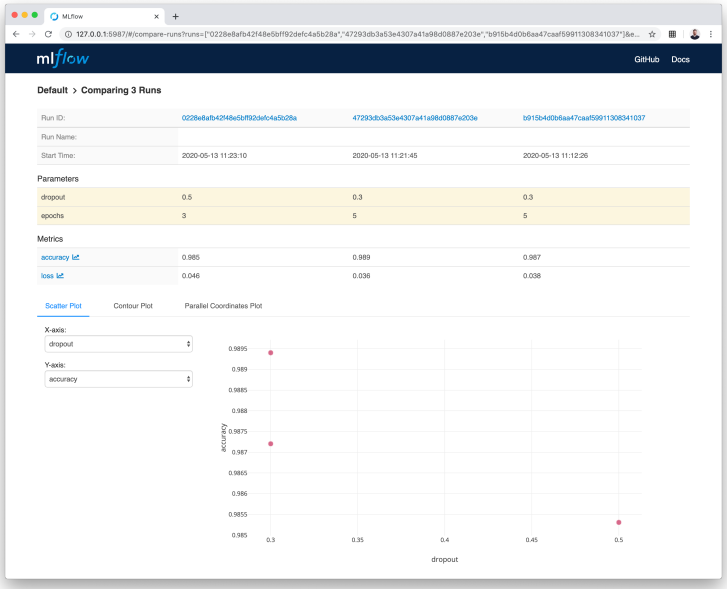
Deep Learning

Gestión del proceso de aprendizaje

mlflow

Lanzar conjuntos de experimentos parametrizados





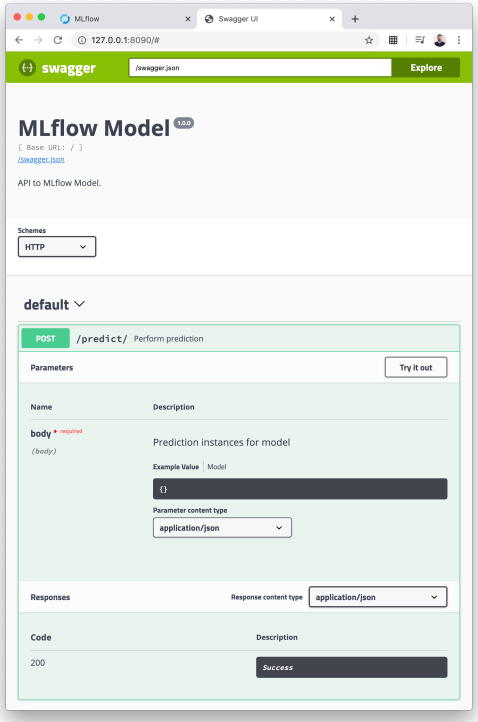
Deep Learning

Gestión del proceso de aprendizaje

Desplegar modelo en REST

mlflow.R

```
# Desplegar el modelo disponible
# http://127.0.0.1:8090/
mlflow_rfunc_serve(
    model_uri="mlruns/0/0228e8afb42f48e5bff92defc4a5b28a/artifacts/model",
    port=8090)
```



Deep Learning

Gestión del proceso de aprendizaje

Alternativas

Weights & Biases — https://www.wandb.com/

Sacred — https://github.com/IDSIA/sacred

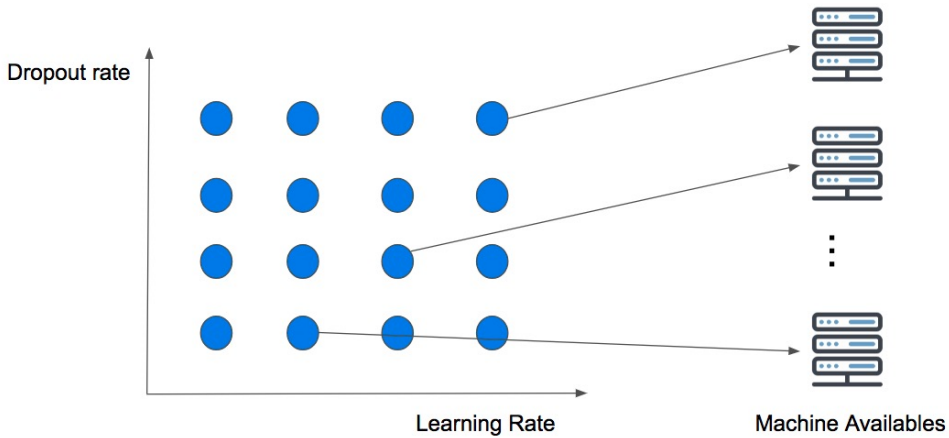
comet — https://www.comet.ml/site/

SIGOPT — https://sigopt.com/product/optimization-engine/

neptune.ai — https://neptune.ai/

Hiperparámetros

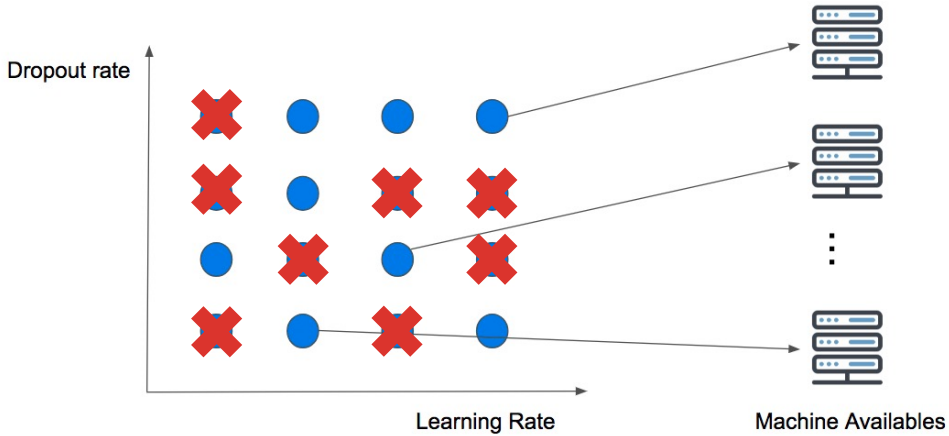
Grid search



A. Gozzoli (2018) *Practical Guide to Hyperparameters Optimization for Deep Learning Models* [link]

Hiperparámetros

Random search



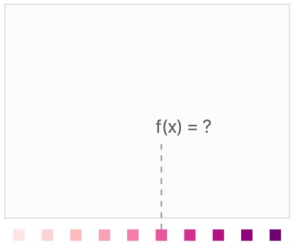
A. Gozzoli (2018) *Practical Guide to Hyperparameters Optimization for Deep Learning Models* [link]

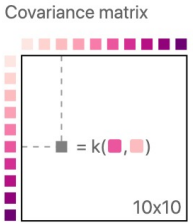
Meta-Learning – AutoML (?)

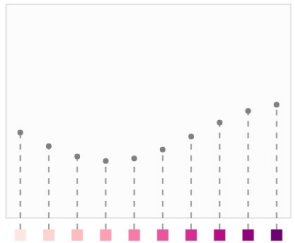
Optimización bayesiana

Se construye un modelo de predicción $P : \vartheta \rightarrow \mathcal{L}$ para estimar el valor que se obtendrá de la métrica a partir de los hiperparámetros

Cálculo de $P(\mathcal{L}|\vartheta)$ como un problema de regresión aplicando un proceso gaussiano







mlflow
https://youtu.be/eEVDIOiYuMk

Keras
Learning rate
Keras tuner [vídeo]

J. Görtler, R. Kehlbeck, O. Deussen (2019) A Visual Exploration of Gaussian Process [link]

Deep Learning

Gestión del proceso de aprendizaje

Meta-Learning

Algoritmos genéticos

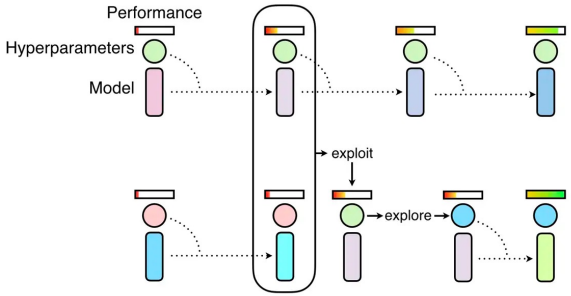
Cada gen codifica los hiperparámetros (+ topología)

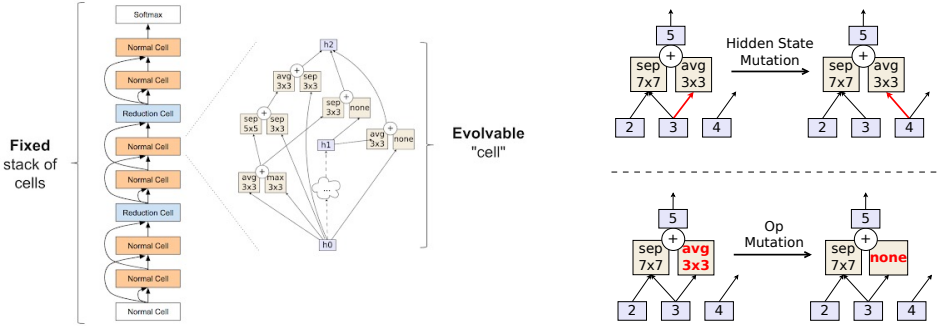
La función de *fitness* es la métrica de aprendizaje

Selección: torneo modificado

Cruce: mezcla de dos redes

Mutación: modificación de una conexión de la red





M. Jaderberg et al. (2017) Population-Based Training of Neural Networks [link]

E. Real et al. (2017) Regularized Evolution for Image Classifier Architecture Search [link]

Meta-Learning

Neural Architecture Search

Resolver el problema de optimización de la topología de la red con una red neuronal

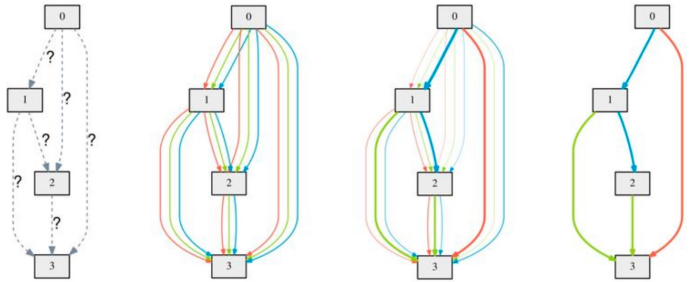
DARTS Differentiable Architecture Search

- Una red se codifica como un conjunto de operaciones encadenadas
- Una operación se encadena con otra si están conectadas (α)
- Algunas de estas operaciones (*dense*, *conv*) utilizan pesos (w)

Objetivo: encontrar los valores α^* que minimizan el error $\mathcal{L}_{val}(w^*, \alpha^*)$, siendo w^* los valores que minimizan el error $\mathcal{L}_{train}(w, \alpha^*)$

Procedimiento problema de optimización bi-nivel

1. Aplicar gradiente descendente sobre el error de entrenamiento $\mathcal{L}_{train}(w_{k-1}, \alpha_{k-1})$ para obtener w_k
2. Con w_k fijo, aplicar gradiente descendente sobre el error de validación $\mathcal{L}_{val}(w_k, \alpha_{k-1})$ para obtener α_k



H. Liu, Y. Yang (2018) DARTS: Differentiable Architecture Search [link]

R. Karim (2019) Illustrated: Efficient Neural Architecture Search [link]

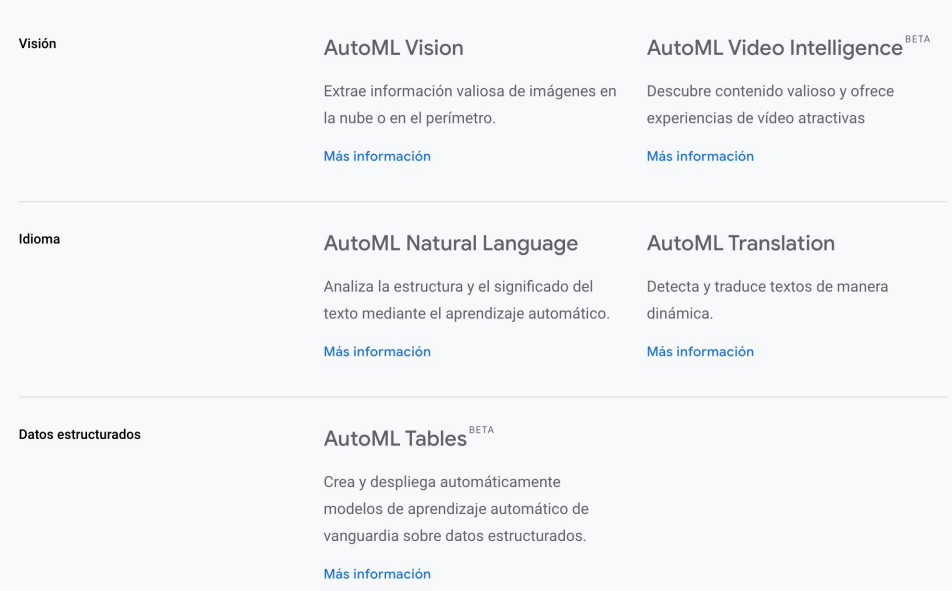
Meta-Learning

Google AutoML

Cloud AutoML es un paquete de productos de aprendizaje automático que permite a los desarrolladores con poca experiencia en la materia preparar modelos de alta calidad adaptados a las necesidades de su negocio. Todo ello gracias a la tecnología de búsqueda con arquitectura neuronal y al aprendizaje por transferencia de última generación de Google.

Combinación de *transfer learning + neural architecture search*

R. Thomas (2018) An opinionated introduction to AutoML and Neural Architecture Search [link]



Deep Learning

Gestión del proceso de aprendizaje

Meta-Learning

.

AutoKeras https://autokeras.com/

Microsoft AutoML https://www.microsoft.com/en-us/research/project/automl/

Amazon SageMaker https://aws.amazon.com/es/sagemaker/

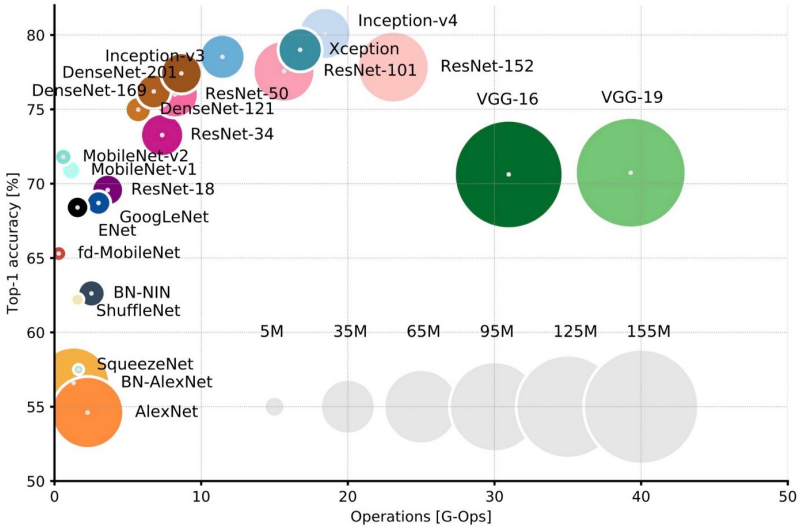
auto-sklearn https://automl.github.io/auto-sklearn/master/

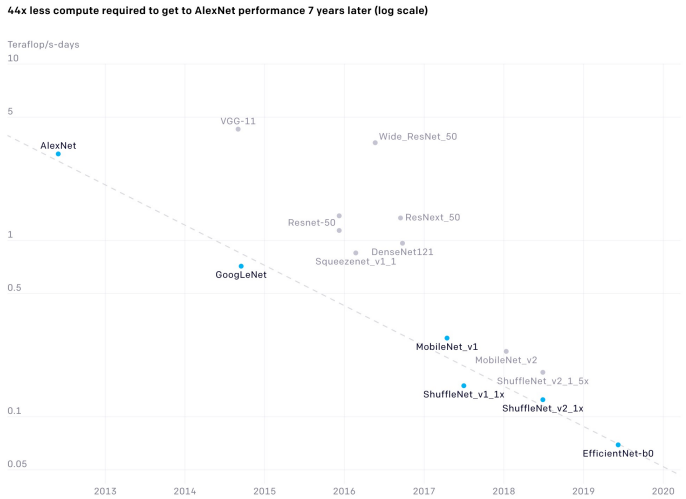
Índice

Deep Learning

Transferencia de aprendizaje

Redes pre-entrenadas





E. Culurciello (2018) Neural Network Architectures [link]

S. Ilic (2020) CNN Architectures - implementations [link]

D. Hernández, T. Brown (2020) AI and Efficiency [link]

Deep Learning

Transferencia de aprendizaje

Redes pre-entrenadas

Topología definida

Entrenadas con un conjunto de datos general, como ImageNet

Consiguen resultados *state-of-the-art* (SOTA)

Keras: Xception, Inception V3, ResNet50, VGG16, VGG19, MobileNet, …

https://keras.rstudio.com/articles/applications.html

https://tensorflow.rstudio.com/guide/tfhub/hub-with-keras/

dogsVScats_transfer.R

```
## Clasificación con red ya entrenada
model_resnet50 <- application_resnet50(
   weights = "imagenet"
)

img_path <- './cat1.jpg'
img <- image_load(img_path, target_size = c(224,224))
x <- image_to_array(img)

x <- array_reshape(x, c(1, dim(x)))
x <- imagenet_preprocess_input(x)

preds <- model_resnet50 %>% predict(x)
imagenet_decode_predictions(preds, top = 3)[[1]]
```

```
  class_name class_description      score
1  n02123394       Persian_cat 0.89344072
2  n02123045             tabby 0.01761619
3  n04589890     window_screen 0.01025248
```

Deep Learning

Transferencia de aprendizaje

Redes pre-entrenadas

Topología definida

Entrenadas con un conjunto de datos general, como ImageNet

Consiguen resultados *state-of-the-art* (SOTA)

Keras: Xception, Inception V3, ResNet50, VGG16, VGG19, MobileNet, …

https://keras.rstudio.com/articles/applications.html

dogsVScats_transfer.R

```
## Clasificación con red ya entrenada
model_resnet50 <- application_resnet50(
   weights = "imagenet"
)

img_path <- './dragon1.jpg'
img <- image_load(img_path, target_size = c(224,224))
x <- image_to_array(img)

x <- array_reshape(x, c(1, dim(x)))
x <- imagenet_preprocess_input(x)

preds <- model_resnet50 %>% predict(x)
imagenet_decode_predictions(preds, top = 3)[[1]]
```

```
  class_name class_description      score
1  n01704323       triceratops 0.55621058
2  n01688243    frilled_lizard 0.05542799
3  n02317335          starfish 0.04102186
```

Deep Learning

Transferencia de aprendizaje

Redes pre-entrenadas

Aproximaciones

Uso directo

Utilizar la red entrenada para un problema igual o similar

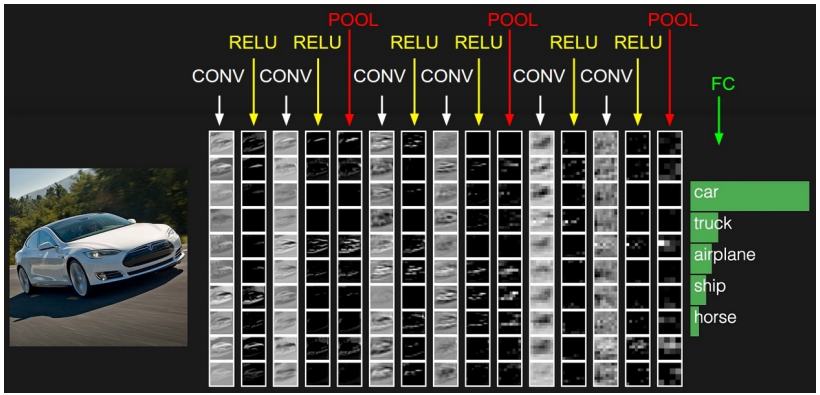
Reutilizar extracción de características

Utilizar las capas convolutivas de la red ya entrenada para extraer características geométricas

Acoplar una nueva subred densa para realizar la clasificación

Fine-tuning

Se re-entrena una parte o todas las capas convolutivas de la red ya entrenada partiendo de los pesos pre-existentes

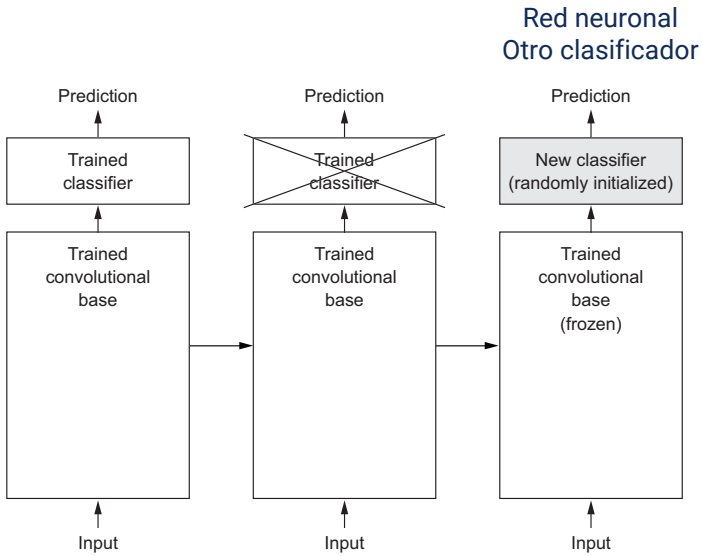


Deep Learning

Transferencia de aprendizaje

Extracción de características

https://keras.rstudio.com/articles/applications.html



dogsVScats_transfer.R

```r
## Extracción de características
# Cargar capa convolutiva de VGG16, pre-entrenada con ImageNet
conv_base <- application_vgg16(
   weights = "imagenet",
   include_top = FALSE,
   input_shape = c(150, 150, 3)
)

# Congelar las capas convolutivas ya entrenada
freeze_weights(conv_base)

# Acoplar nuevo clasificador (red densa)
model <- keras_model_sequential() %>%
   conv_base %>%
   layer_flatten() %>%
   layer_dense(units = 512, activation = "relu") %>%
   layer_dense(units = 1, activation = "sigmoid")

# Entrenar modelo
model %>% compile(
   optimizer = optimizer_rmsprop(lr = 2e-5),
   loss = "binary_crossentropy",
   metrics = c("accuracy")
)

history <- model %>%
   fit_generator(
      train_data,
      steps_per_epoch = 100,
      epochs = 2, # 30
      validation_data = validation_data,
      validation_steps = 50
   )
```

```
Test
$loss
[1] 0.33582

$accuracy
[1] 0.858
```

Deep Learning

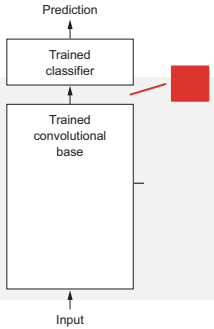
Transferencia de aprendizaje

Extracción de características

https://keras.rstudio.com/articles/applications.html

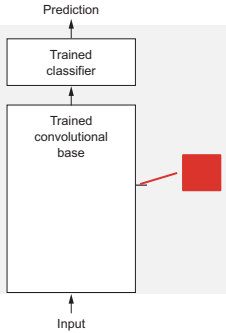
Opción 1: Seleccionar solo parte convolutiva

```
model <- application_vgg16(weights = 'imagenet', include_top = FALSE)

img_path <- "elephant.jpg"
img <- image_load(img_path, target_size = c(224,224))
x <- image_to_array(img)
x <- array_reshape(x, c(1, dim(x)))
x <- imagenet_preprocess_input(x)

features <- model %>% predict(x)
```



Opción 2: Truncar una parte de la red

```
base_model <- application_vgg19(weights = 'imagenet')
model <- keras_model(inputs  = base_model$input,
                     outputs = get_layer(base_model, 'block4_pool')$output)

img_path <- "elephant.jpg"
img <- image_load(img_path, target_size = c(224,224))
x <- image_to_array(img)
x <- array_reshape(x, c(1, dim(x)))
x <- imagenet_preprocess_input(x)

block4_pool_features <- model %>% predict(x)
```

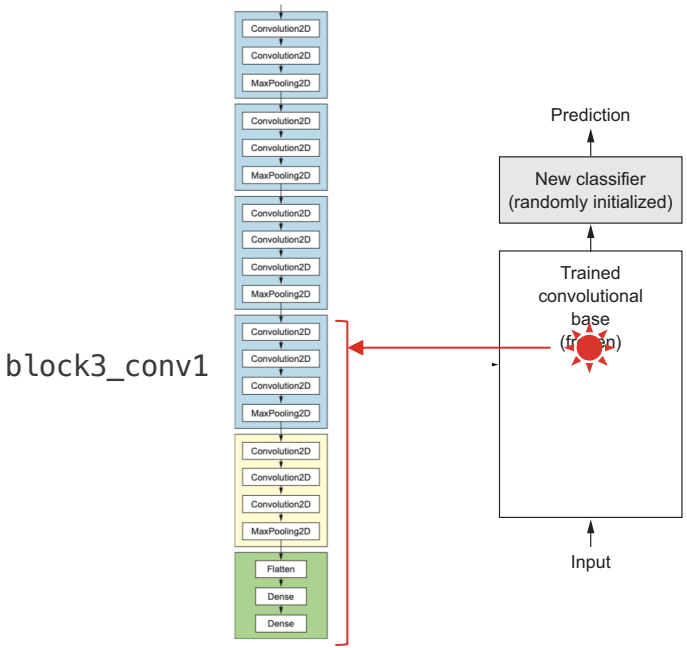


Deep Learning

Transferencia de aprendizaje

Fine tuning

https://keras.rstudio.com/articles/applications.html



1. Añadir capa FC personalizada
2. Congelar los pesos de la red base
3. Entrenar la capa FC
4. Descongelar capas de la red base
5. Entrenar capa descongelada y FC

dogsVScats_transfer.R

```
## (continúa ejemplo anterior)

# 4. Descongelar una parte de la la capa base
unfreeze_weights(conv_base, from = "block3_conv1")

# 5. Entrenar capa descongelada y FC
model %>% compile(
   loss = "binary_crossentropy",
   optimizer = optimizer_rmsprop(lr = 1e-5),
   metrics = c("accuracy")
)

history <- model %>%
   fit_generator(
   train_data,
   steps_per_epoch = 100,
   epochs = 25,
   validation_data = validation_data,
   validation_steps = 50
)
```

```
Test
$loss
[1] 0.3522196

$accuracy
[1] 0.953
```

Índice

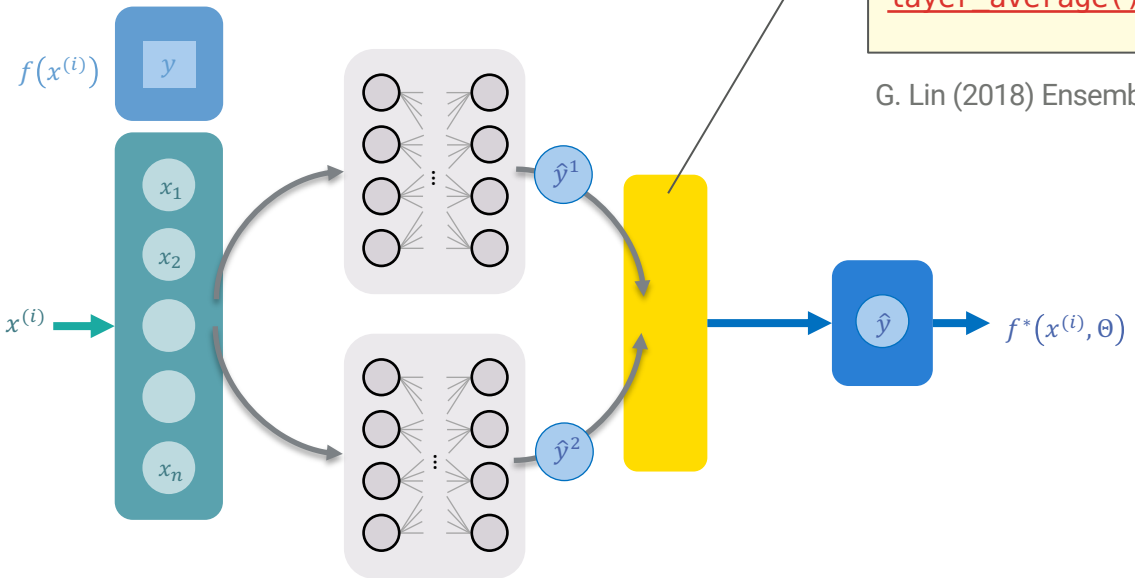
Deep Learning

Ensembles

Bagging
`layer_maximum()`

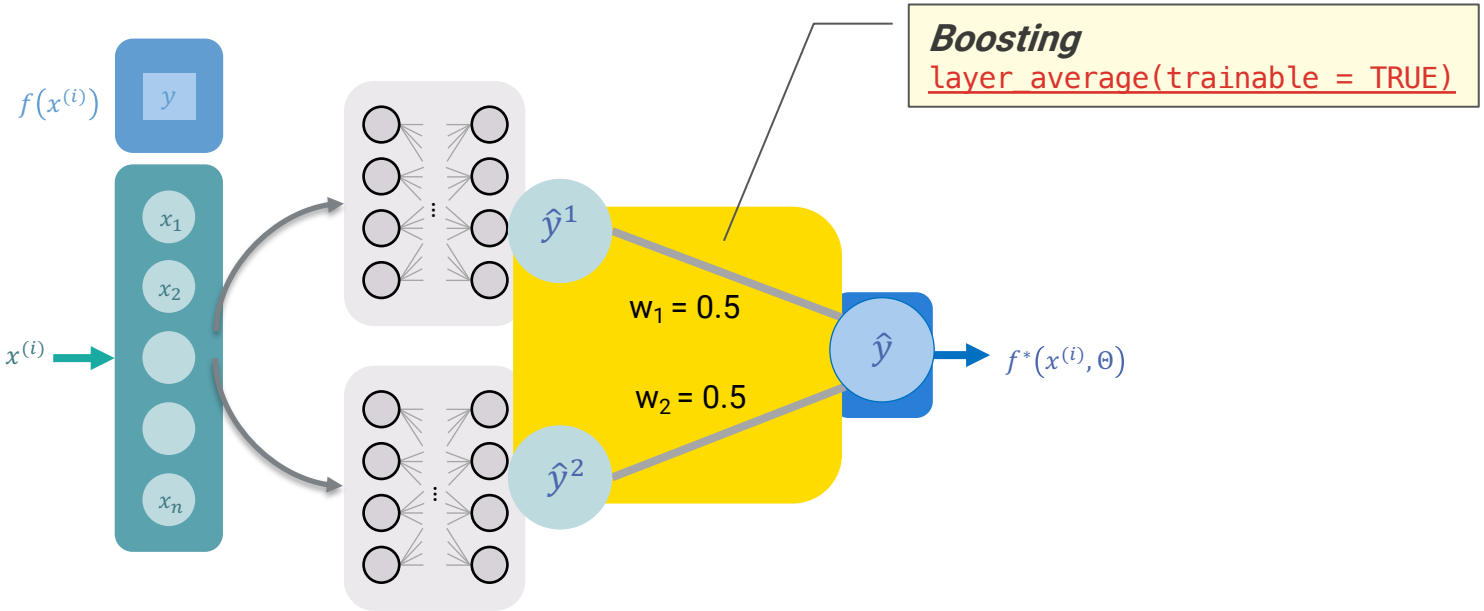
Media
`layer_average()`

G. Lin (2018) Ensembling Keras models in R [link]



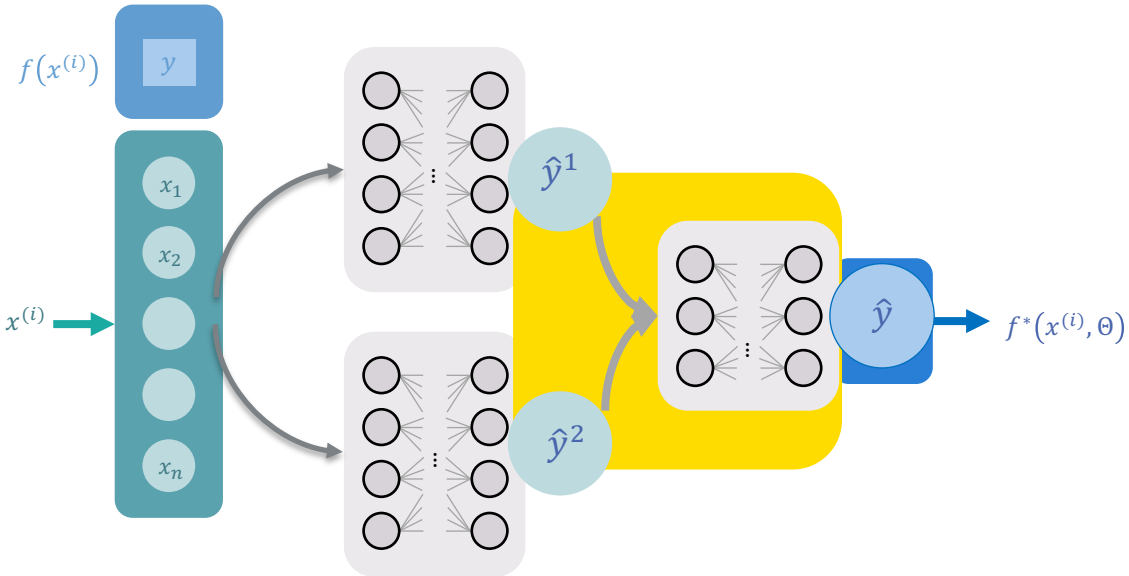
Deep Learning

Ensembles



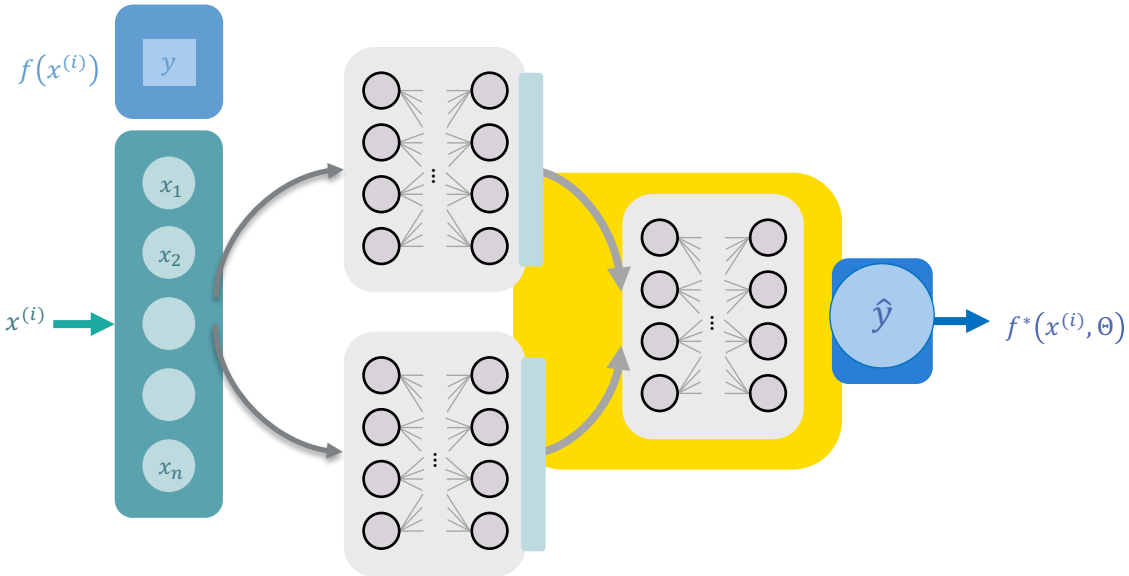
Deep Learning

Ensembles



Deep Learning

Ensembles



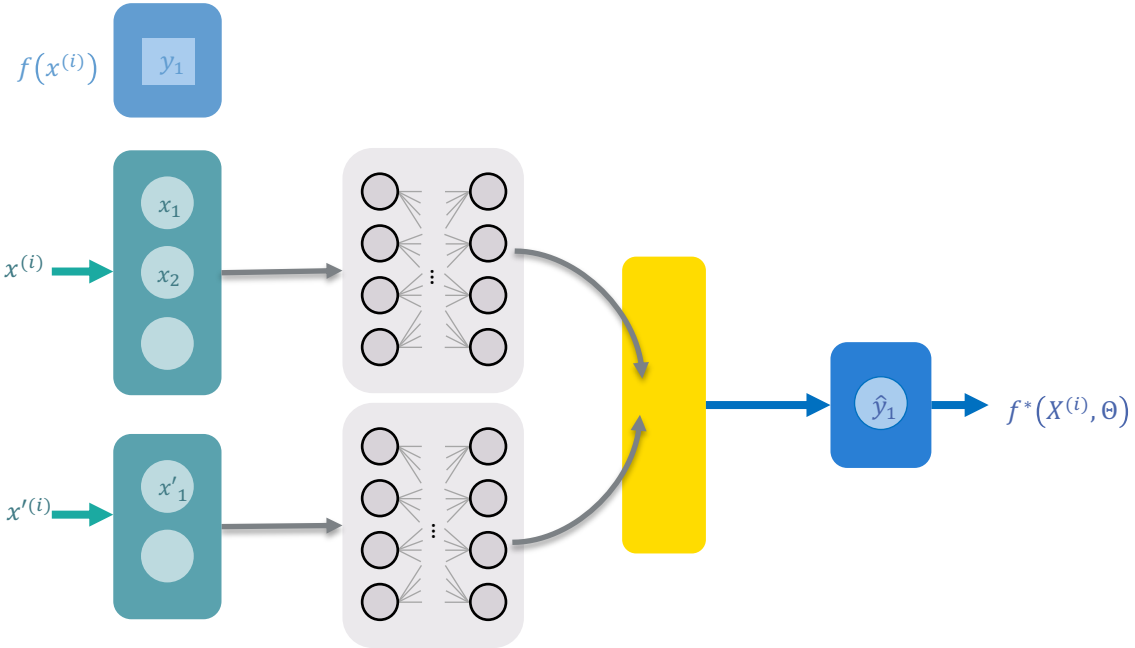
Deep Learning

Temas avanzados ► Ensamblado de modelos

Ensembles

Pickles
Female, 5 Mths, Domestic Short Hair
Kuala Lumpur, by allergicresc..

Pickles was rescued off the street in really bad shape. Her mange and fungus is so bad that she's actually bald in most of her body. She also has a bad case of flu. Luckily, with the right medication, treatment, nutritious food an..

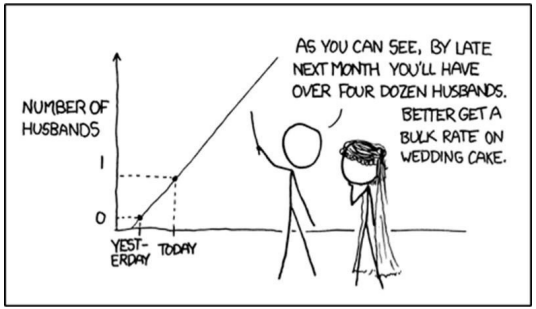


Series de datos

Secuencias de datos con relación temporal entre ellos

Univariate
Una variable a la entrada (una a la salida)

Multivariate
Varias variables a la entrada (una o varias a la salida)



Extrapolating https://xkcd.com/605/

Características del aprendizaje

Se busca predecir un valor (o valores) de la serie en el futuro

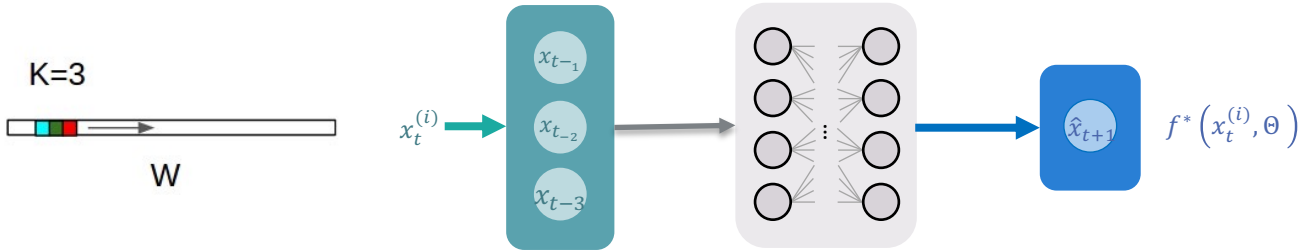
La predicción se realiza a partir de los valores *t*, *t*−*1*, *t*−*2*, ... de la(s) secuencia(s)

Deep Learning

Temas avanzados ▶ Series de datos

MLP “temporal”

La secuencia se segmenta en intervalos



Convolución temporal

Se aplica una convolución sobre la secuencia para agregar los valores

`layer_conv_1d`

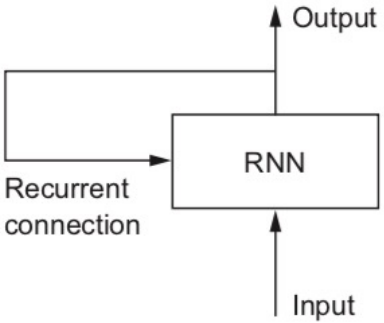
https://stackoverflow.com/questions/42883547/intuitive-understanding-of-1d-2d-and-3d-convolutions-in-convolutional-neural-n

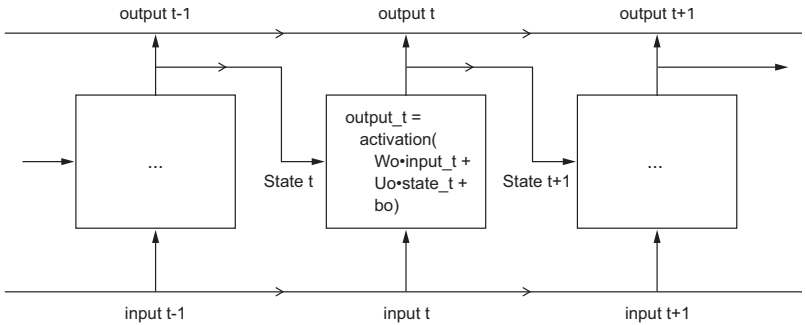
Deep Learning

Temas avanzados ▶ Series de datos

Redes neuronales recurrentes

La salida depende de los valores de entrada y de la salida previa de la red





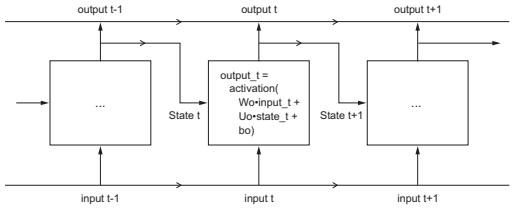
`layer_simple_rnn`

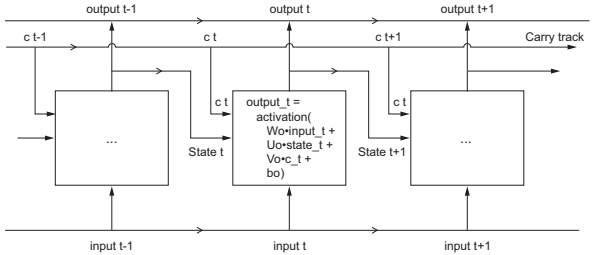
Deep Learning

Temas avanzados ▶ Series de datos

Redes neuronales recurrentes avanzadas

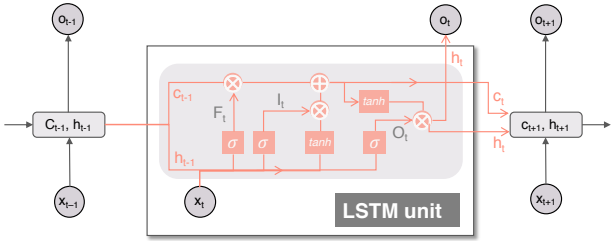
Arquitecturas más complejas para conseguir recordar a más largo plazo





LSTM: Long-short therm memory

`layer_lstm`



C. Olah (2015) *Understanding LSTM Networks* [link]

Deep Learning

Lenguaje natural

Tipo especial de secuencia de datos

`fakenews-text.R`

Características del aprendizaje

Codificación de los datos

> *Embeddings*

Contexto hacia delante y hacia atrás

> Mecanismos de atención

Múltiples tareas: extracción de entidades y relaciones, traducción automática, conversión imagen-texto, etc.

> Modelos de tipo *sequence-to-sequence*

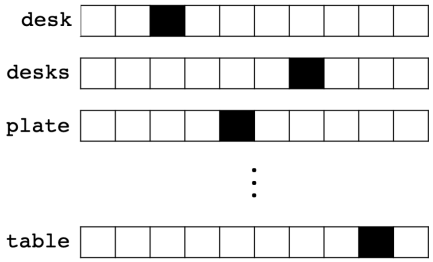
> Modelos de tipo *transformer*

E. Saravia, S. Poria (2020) Modern Deep Learning Techniques Applied to Natural Language Processing [link]

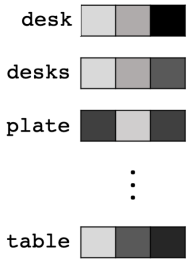
Embeddings

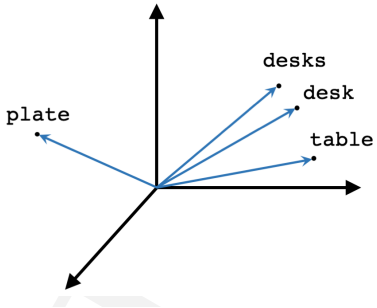
Representación de los elementos del lenguaje mediante vectores numéricos

Un elemento del lenguaje (por ejemplo, una palabra) se representa con un conjunto de vectores (embeddings), cada uno de ellos codificando una característica (feature). Estos valores codificados son los que se utilizan como entrada de la red neuronal



one-hot encoding





vector space encoding

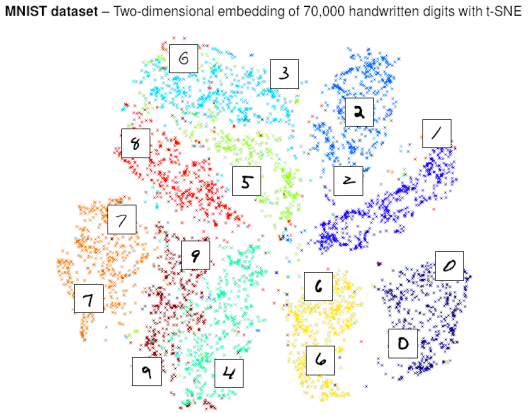
M.T. Pilehvar, J. Camacho Collados (2020) Embeddings in Natural Language Processing. Morgan & Claypool [link]

Deep Learning

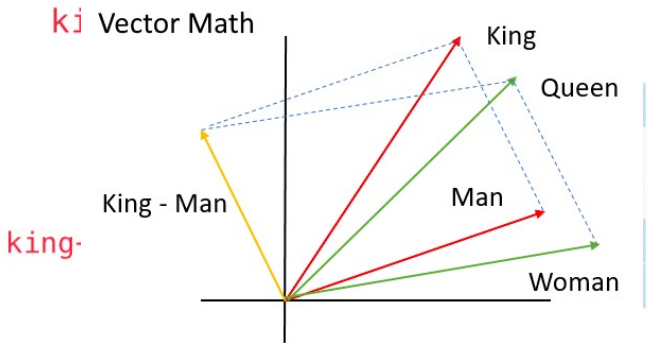
Embeddings

Dos elementos del lenguaje similares tendrán una representación cercana en el espacio vectorial definido por los embeddings

Posible operar con los vectores aplicando su semántica: <rey> - <hombre> + <mujer> = <reina>



A. Fabisch (2020) t-Distributed Stochastic Neighbor Embedding (t-SNE) [link]

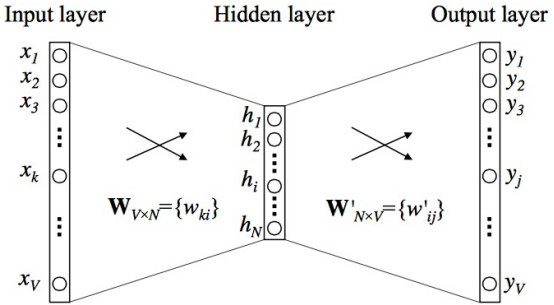


J. Alammar (2020) The Illustrated Word2vec [link]

Deep Learning

Embeddings

Se calculan a partir de sus propiedades lingüísticas, información contextual y recursos lingüísticos adicionales



X. Rong (2014) word2vec Parameter Learning Explained [link]

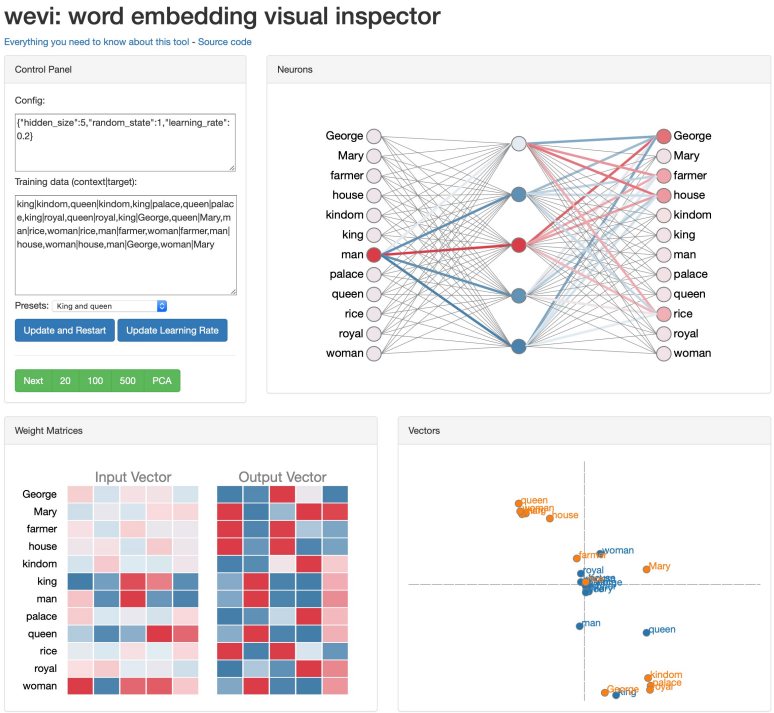
Embeddings (pre-entrenados): word2vec, GloVe, fasttext, etc.

Modelos de lenguaje (pre-entrenados): BERT (Google), XLNet (Google), GPT-3 (OpenAI)

```
layer_embedding
```

Deep Learning

Temas avanzados ► Procesamiento del Lenguaje Natural



X. Rong (2014) Wevi: word embedding visual inspector [link]

Arquitecturas avanzadas

Redes recurrentes bidireccionales

Consideran el contexto por delante y por detrás de una palabra

Mecanismos de atención

Sustituyen las recurrencias por “convoluciones” aplicadas a representaciones matriciales de texto para fijarse en partes concretas de las frases

Transformers

Arquitecturas con secciones de codificación-decodificación

Modelos predictivos de lenguaje (pre-entrenados):

BERT (Google), XLNet (Google), GPT-3 (OpenAI)

Deep Learning

Tareas

General Language Understanding Evaluation (2019)

clasificación, similitud, predicción, etc.

GLUE

https://youtu.be/otvqkWFvUZU

Probando a GPT-3

Opinion

GPT-3, Bloviator: OpenAI's language generator has no idea what it's talking about

Tests show that the popular AI still has a poor grasp of reality.

by **Gary Marcus** and **Ernest Davis** August 22, 2020

https://www.technologyreview.com/2020/08/22/1007539/gpt3-openai-language-generator-artificial-intelligence-ai-opinion/

Deep Learning

Herramientas

.

spaCy https://spacy.io/

HUGGING FACE https://huggingface.co/

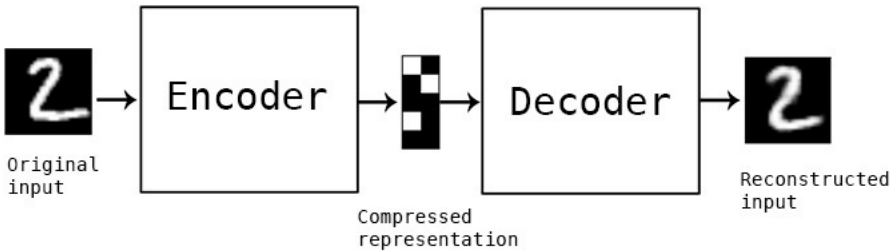
AllenNLP https://allennlp.org/

NLTK https://www.nltk.org/

Deep Learning

Temas avanzados ► *Autoencoders & transformers*

Autoencoder : modelo optimizado para obtener una réplica de la entrada

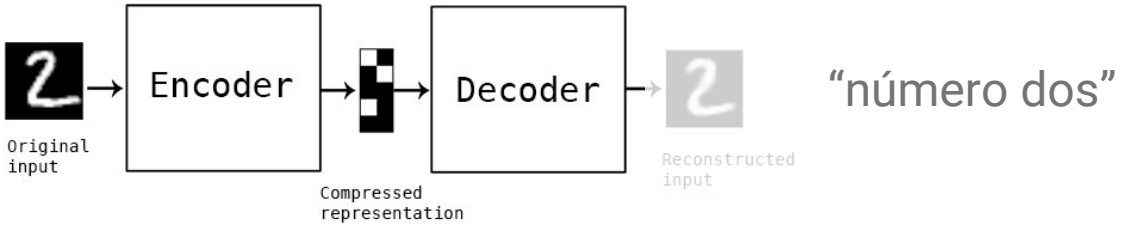


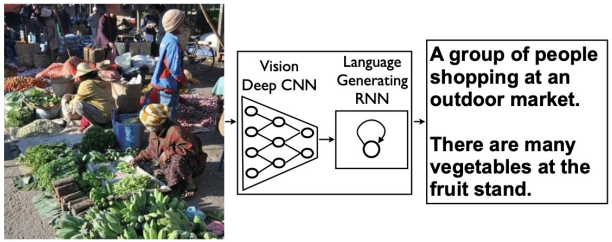
F. Chollet (2016) Building autoencoders with Keras [link]

- Dos módulos: codificador, decodificador
- Representación intermedia en espacio latente
- Aprendizaje auto-supervisado (*self-supervised*)
- Entrenamiento: minimizar $x - D(E(x))$

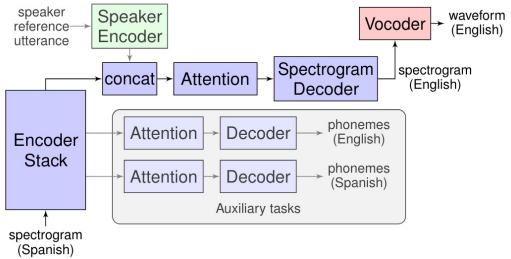
Deep Learning

Sequence-to-sequence: modelo optimizado para obtener una secuencia a partir de una secuencia





O. Vinyals, A. Toshev, S. Bengio, D. Erhan (2014) Show and Tell: A Neural Image Caption Generator [link]

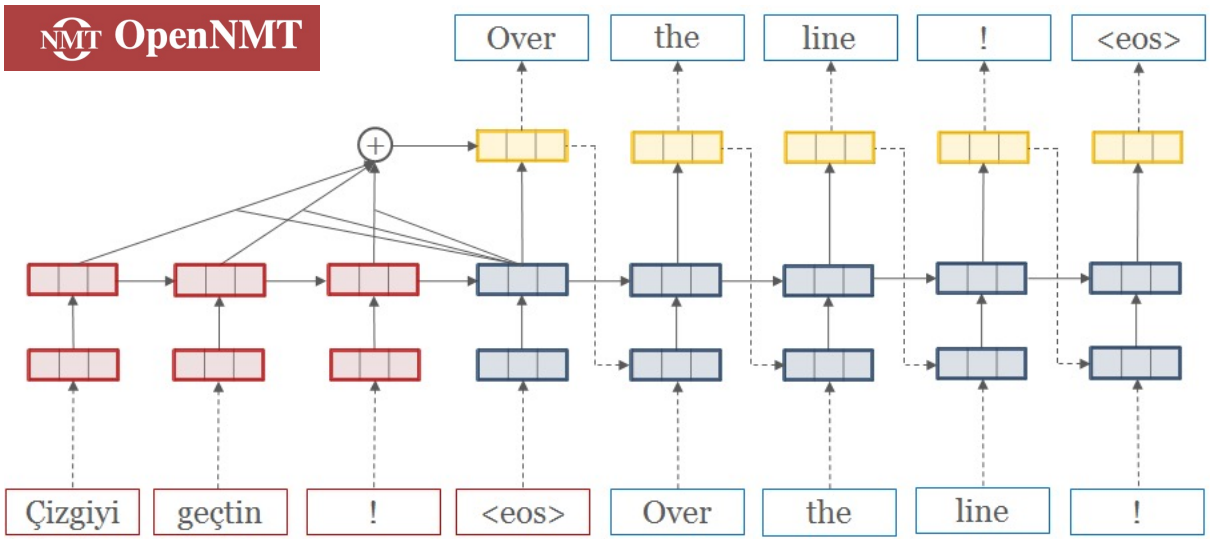


Y. Jia et al (2019) Direct speech-to-speech translation with a sequence-to-sequence model [link]

Deep Learning

Temas avanzados ► *Autoencoders & transformers*

Traducción automática

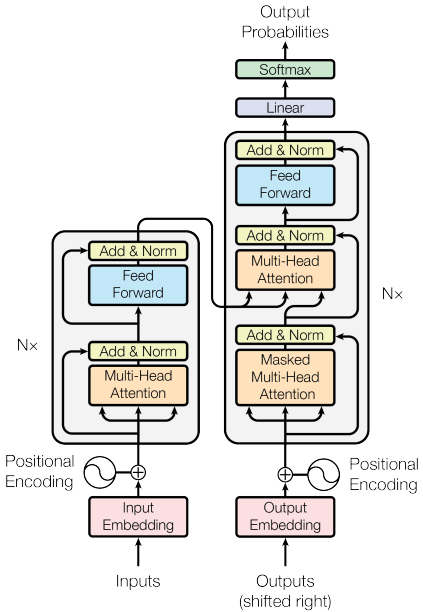


L. Gatys, A. Ecker, M. Bethge (2016) Machine Translation Reading List [link]

Deep Learning

Temas avanzados ► *Autoencoders & transformers*

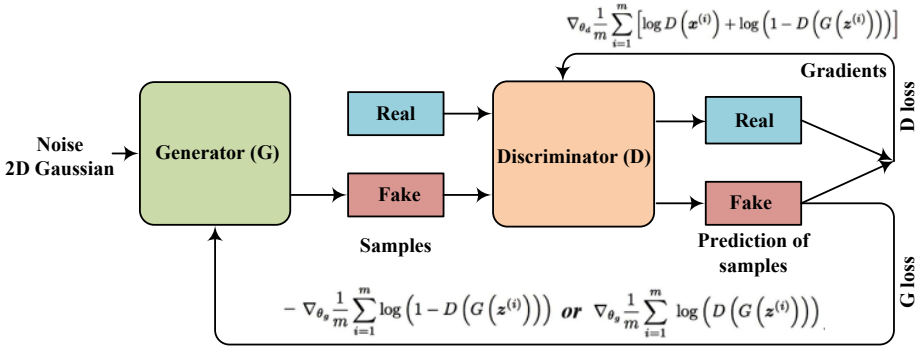
Transformer: modelo para sequence-to-sequence que sólo utiliza atención



A. Vaswani et al. (2017) Attention is all you need [link]

Deep Learning

Generative adversarial networks (GANs): procedimiento para entrenamiento de modelos generativos



I. Goodfellow et al. (2014) Generative Adversarial Nets. NIPS 2014 [link]

D. Saxena, J. Cao (2020) Generative Adversarial Networks (GANs): Challenges, Solutions, and Future Directions [link]

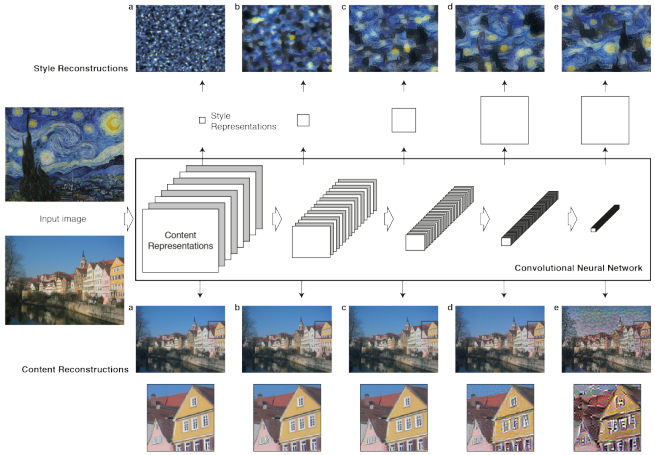
T. Karras et al. (2020) StyleGAN2 [link]

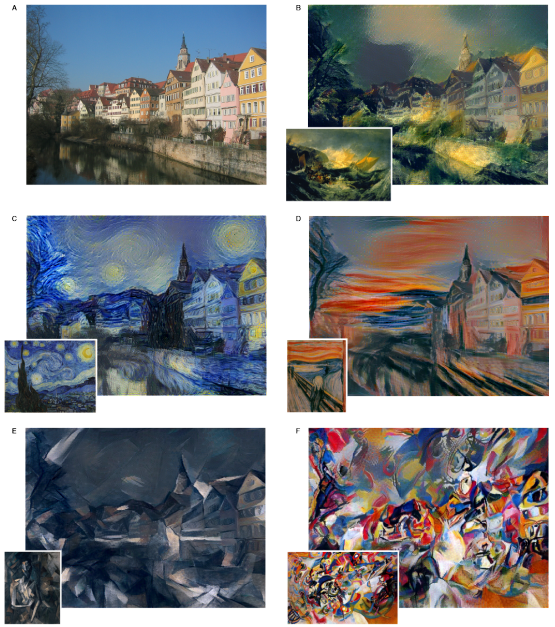
Arxiv Insights (2019) Face editing with Generative Adversarial Networks [link]

Deep Learning

Temas avanzados ▶ Modelos generativos

Transferencia de estilo





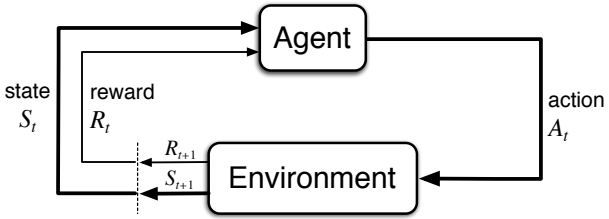
L. Gatys, A. Ecker, M. Bethge (2016) A Neural Algorithm of Artistic Style [link]

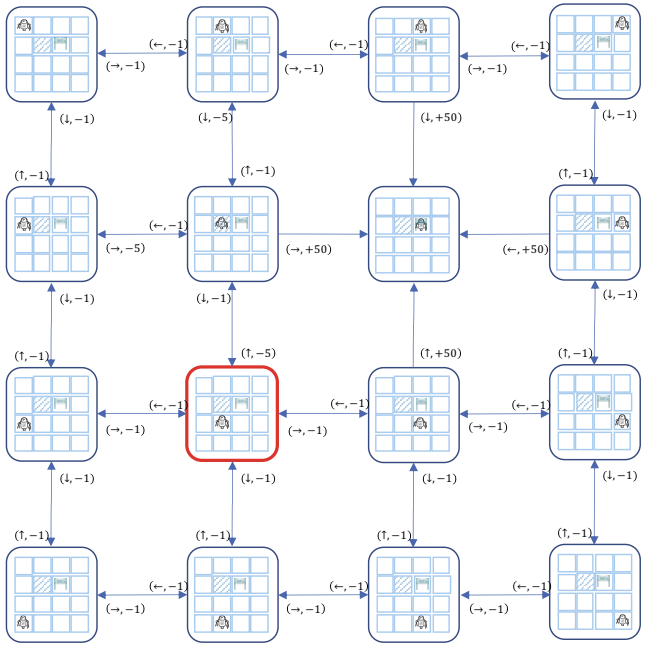
Índice

Aprendizaje profundo por refuerzo

Concepto

Aprendizaje por refuerzo: Resolución de una tarea mediante “prueba y error”

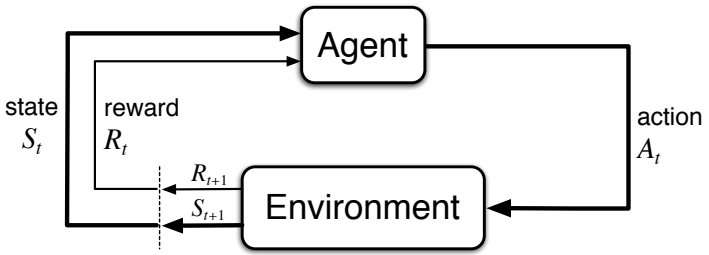




R.S. Sutton, A.G. Barto (2018)
Reinforcement Learning. MIT Press.

Aprendizaje profundo por refuerzo

Formulación



Markov Decision Process - MDP

tiempo:	$t = 0, 1, 2, \ldots$
estado:	$S_t \in \mathcal{S}$
acción:	$A_t \in \mathcal{A}(S_t)$
recompensa:	$R_{t+1} \in \mathcal{R} \subset \mathbb{R}$
estado siguiente:	$S_{t+1} \in \mathcal{S}$

secuencia:

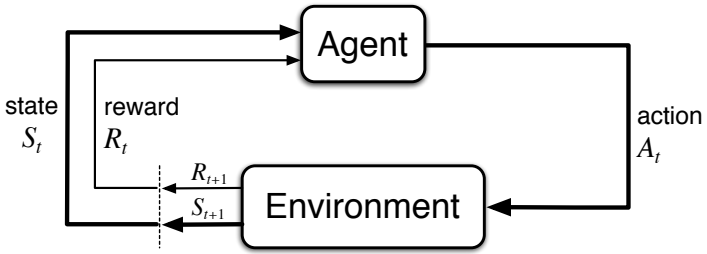
$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \ldots$

dinámica del MDP:

$p(s', r|s, a) \doteq Pr\{S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\}$

Aprendizaje profundo por refuerzo

Formulación



Markov Decision Process - MDP

Recompensa acumulada:

$$G_t \doteq R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T$$

Recompensa con descuento:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots = \sum_{k=0}^{T} \gamma^k R_{t+k+1}$$

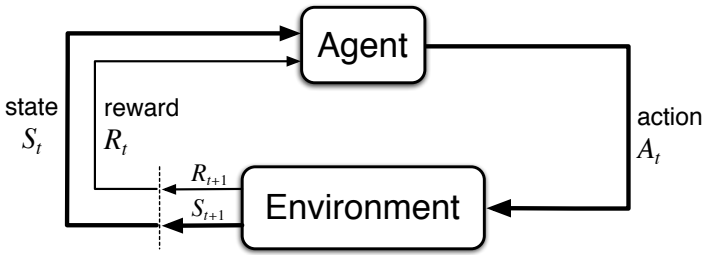
Política de actuación: π

Acción según π : $\pi(a|s)$

Política de actuación óptima: π_*

Aprendizaje profundo por refuerzo

Formulación



Markov Decision Process - MDP

Función de estado-valor [*state value*] (para todo s):

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s\right]$$

Función de acción-valor *[action value]* (para todo (s, a)):

$$q_\pi(s,a) \doteq \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \,\middle|\, S_t = s, A_t = a\right]$$

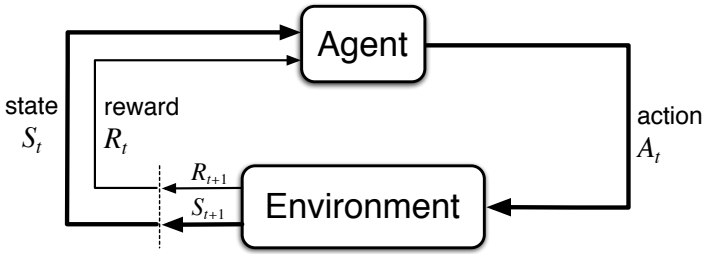
Comparación de políticas de actuación y política óptima

$\pi \geq \pi' \Leftrightarrow v_\pi(s) \geq v'_\pi(s)$

$\pi_* \geq \pi' \; \forall \pi'$ *(garantizada)*

Aprendizaje profundo por refuerzo

Formulación



Ecuación de Bellman (recursividad de v_π)

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t \mid S_t = s]$$

$$= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} \mid S_t = s]$$

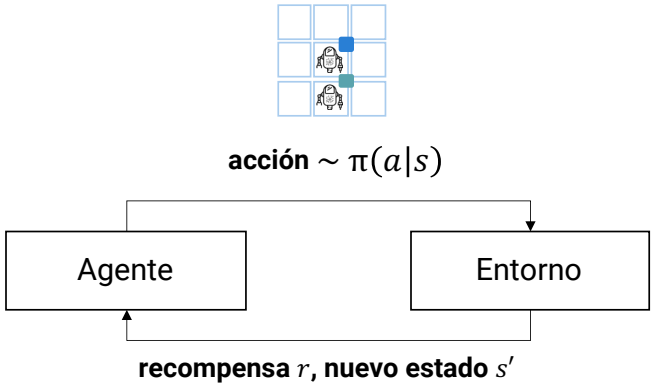
$$= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s]$$

Ecuación de Bellman (optimalidad de v_{π_*})

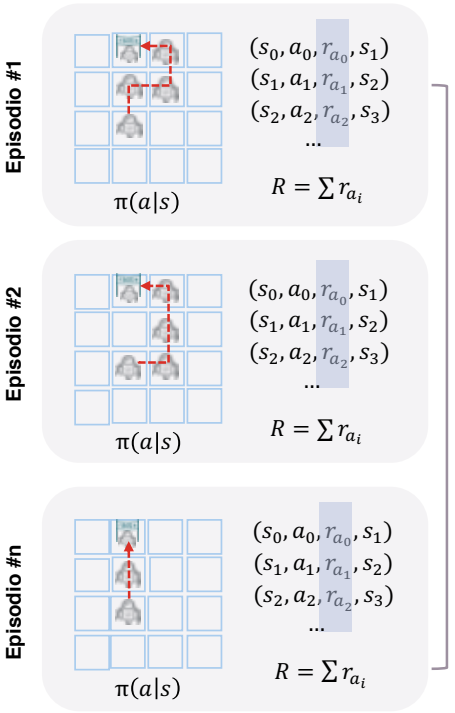
$$v_{\pi_*}(s) = \max_{a \in \mathcal{A}(s)} q_{\pi_*}(s, a)$$

Aprendizaje profundo por refuerzo

Resolución



Episodios de entrenamiento



Estado	Acción	q
(3, 0)	↑	10
(2, 0)	→	12
...		...

training

$$\pi' = \arg\max_{\pi} \sum_{t=0} \gamma^t * r_{a_t}(s_t, s_{t+1})$$

con $a_t \sim \pi(a|s)$

$\gamma \in [0, 1]$: tasa de descuento

Aprendizaje profundo por refuerzo

Resolución

Método de Montecarlo (MC)

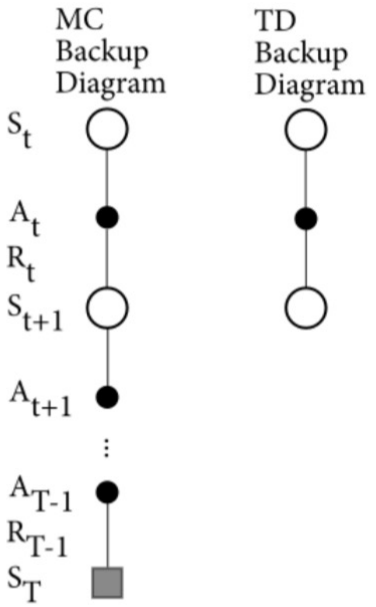
Estimar la tabla Q iterativamente con los datos recogidos en cada episodio con la política π

$$Q'(s,a) \approx Q(s,a) + \hat{q}_{\pi_i}(s,a)$$

Q-Learning (TD)

Estimar la tabla Q iterativamente con los datos recogidos en cada paso con la política π

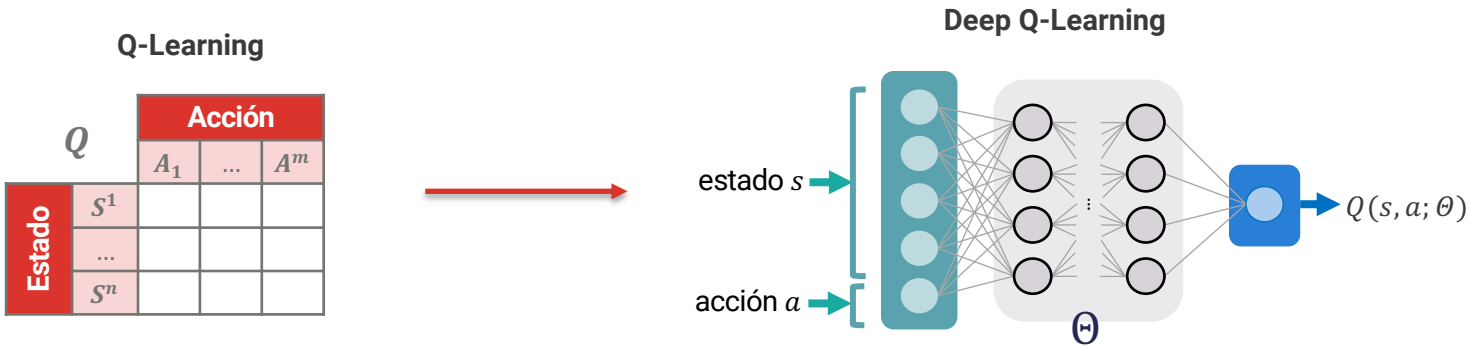
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \max_{a \in \mathcal{A}} Q(S_{t+1}, a) - Q(S_t, A_t)\right)$$



Aprendizaje profundo por refuerzo

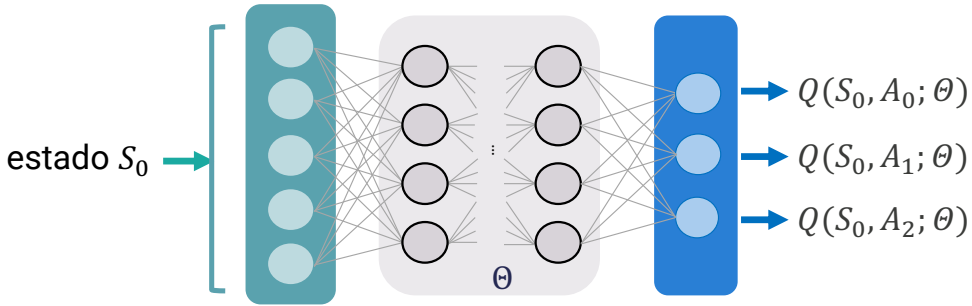
Resolución

Aprendizaje profundo por refuerzo: Uso de redes neuronales para estimar la “mejor acción”



Aprendizaje profundo por refuerzo

DQN – Deep Q-Network



Entrenamiento *(learning)*

Entrada: S

Salida: $Q(S;\Theta)$

Función de error: $\mathcal{L}(Q(S), Q(S;\Theta))$

¡No disponemos de esta salida!

Guardar una base de datos o memoria de *experiencias*

Se puede utilizar en cualquier orden y más de una vez

Pero podemos estimarla a partir de:

$S_0\ A_0\ R_1\ S_1$

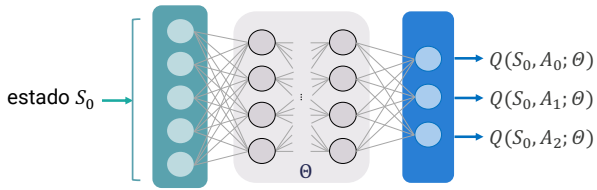
$$Q(S_0, A_0) \leftarrow Q(S_0, A_0) + \alpha\left(R_1 + \gamma \max_{a \in \mathcal{A}} Q(S_1, a) - Q(S_0, A_0)\right)$$

Aprendizaje profundo por refuerzo

DQN – Deep Q-Network

$\ell\left(x^{(i)}, \Theta\right) = \mathcal{L}(Q(S), Q(S, A; \Theta))$

Con MSE: $\mathrm{E} = \mathbb{E}_{\pi}\left[\left(Q(S, A) - Q(S, A; \Theta)\right)^2\right]$



Recordemos que: $\Delta\Theta = \Delta w = -\eta \frac{\partial E}{\partial w}$

Desarrollamos: $\frac{\partial E}{\partial w} = 2\left(Q(S, A) - Q(S, A; \Theta)\right) \frac{\partial Q(S, A; \Theta)}{\partial w}$

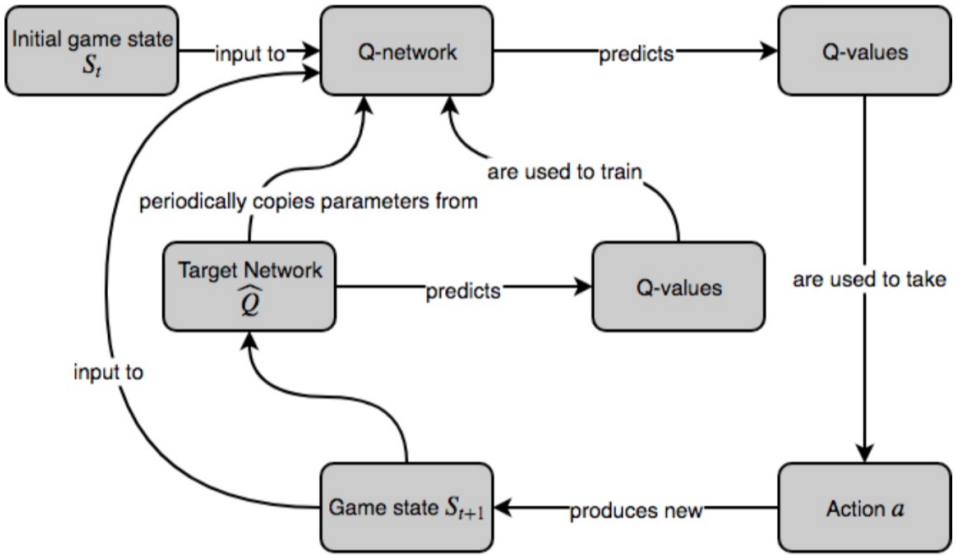
Sustituimos: $\Delta\Theta = -2\eta \left(Q(S, A) - Q(S, A; \Theta)\right) \nabla_{\Theta} Q(S, A; \Theta)$

Simplificamos y aplicamos la expresión de Q-Learning:

$$\Delta\Theta = \alpha \left(R + \gamma \max_{a \in \mathcal{A}} Q(S', a; \Theta) - Q(S, A; \Theta) \right) \nabla_{\Theta} Q(S, A; \Theta)$$

Aprendizaje profundo por refuerzo

DQN – Deep Q-Network



A. Zai, B. Brown (2018) **Deep Reinforcement Learning in Action**. Manning.

Aprendizaje profundo por refuerzo

DQN – Deep Q-Network

V. Mnih et al. (2015) Human-level control through deep reinforcement learning. Nature 518, 529-533
Deep Mind DQN (2016). Más: https://deepmind.com/research/dqn/

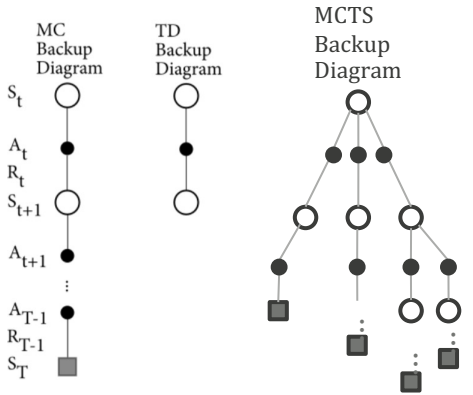
Aprendizaje profundo por refuerzo

Monte Carlo Tree Search

AlphaGo

Resolución de juegos mediante árboles de búsqueda

Aprendizaje profundo por refuerzo



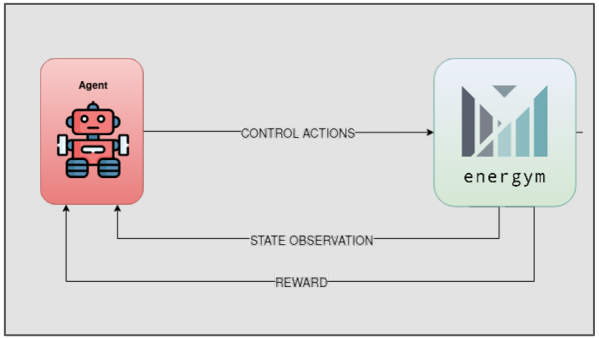
http://www.alphagomovie.com/

D. Silver et al. (2016) Mastering the game of Go with Deep Neural Networks & Tree Search. Nature 529, 484-489

Aprendizaje profundo por refuerzo

Aplicaciones

Alejandro Campoy Nieves (2020) Aprendizaje por refuerzo profundo para la resolución de juegos



energym (2021) An open-source gym-like environment for building energy control

Ángel Murcia Díaz (2021) Aprendizaje profundo por refuerzo en el entorno Google Football

L2RPN (2021) Learning to operate a power grid

Sistemas Inteligentes para Gestión en la Empresa

Máster en Ingeniería Informática

¡Gracias!

Dale a *like* (en la encuesta)