HERRAMIENTAS MODERNAS EN REDES NEURONALES: LA LIBRERÍA TENSORFLOW DE GOOGLE

Antonio Mejías Gil

14 de junio de 2017

Contenido

Introducción y objetivos

- Redes neuronales profundas (DNNs)
  - Redes neuronales artificiales (1940s) con varias capas ocultas
  - Ahora posibilidad real gracias a: **hardware**, **técnicas** y **datos**
  - Nueva moda de *machine learning*: Google, Facebook, etc.

- Objetivos
  1. Conocer los fundamentos de las redes profundas (*convnets* en particular)
  2. Entender el funcionamiento de **TensorFlow**
  3. Aplicar **TensorFlow** a un problema real

Recorrido de TensorFlow

- Librería de Google abierta al público hace 1,5 años
- Un año después: versión 1.0 y *DevSummit*
- Actualmente usado por más 6000 proyectos de software libre

Grafo de operaciones

Grafo TensorFlow

El *grafo de flujo de datos* está compuesto de nodos (**ops**) que operan con **tensores**.

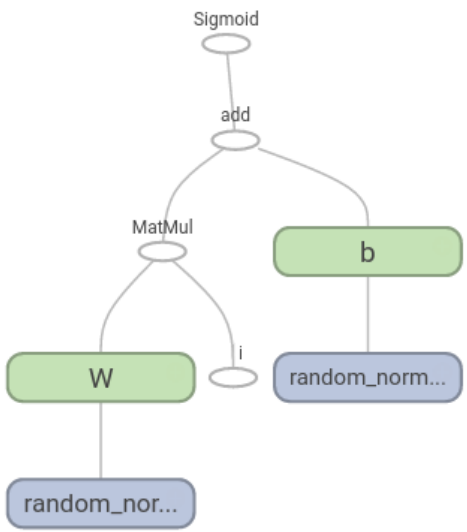
```
import tensorflow as tf


W = tf.Variable(tf.random_normal((3, 4)), name="W")

b = tf.Variable(tf.random_normal((3, 1)), name="b")

i = tf.placeholder(tf.float32, (4, 1), name="i")

o = tf.sigmoid(tf.matmul(W, i) + b)
```



Sesiones de ejecución

Sesiones TensorFlow

Permiten calcular el valor de ciertos nodos (**fetches**) a partir de unos valores de entrada (**feeds**).

- 1 grafo $\Leftrightarrow$ n sesiones
- Uso:

```
session_name.run([lista_outputs], feed_dict={diccio_inputs})

op_name.run(feed_dict=None, session=None)

tensor_name.eval(feed_dict=None, session=None)
```

Variables

Variables TensorFlow

Las variables son nodos que no pierden su valor en los cambios de contexto entre TensorFlow y el programa principal.

Gestión de variables:

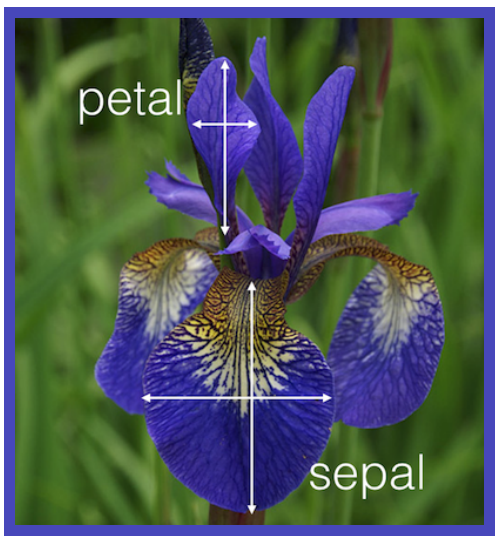
1. Creación de la variable: 3 nuevos nodos
2. Inclusión automática en la colección

   `GraphKeys.TRAINABLE_VARIABLES`

3. Dentro de la sesión, ejecutar nodo:

   `tf.global_variables_initializer`

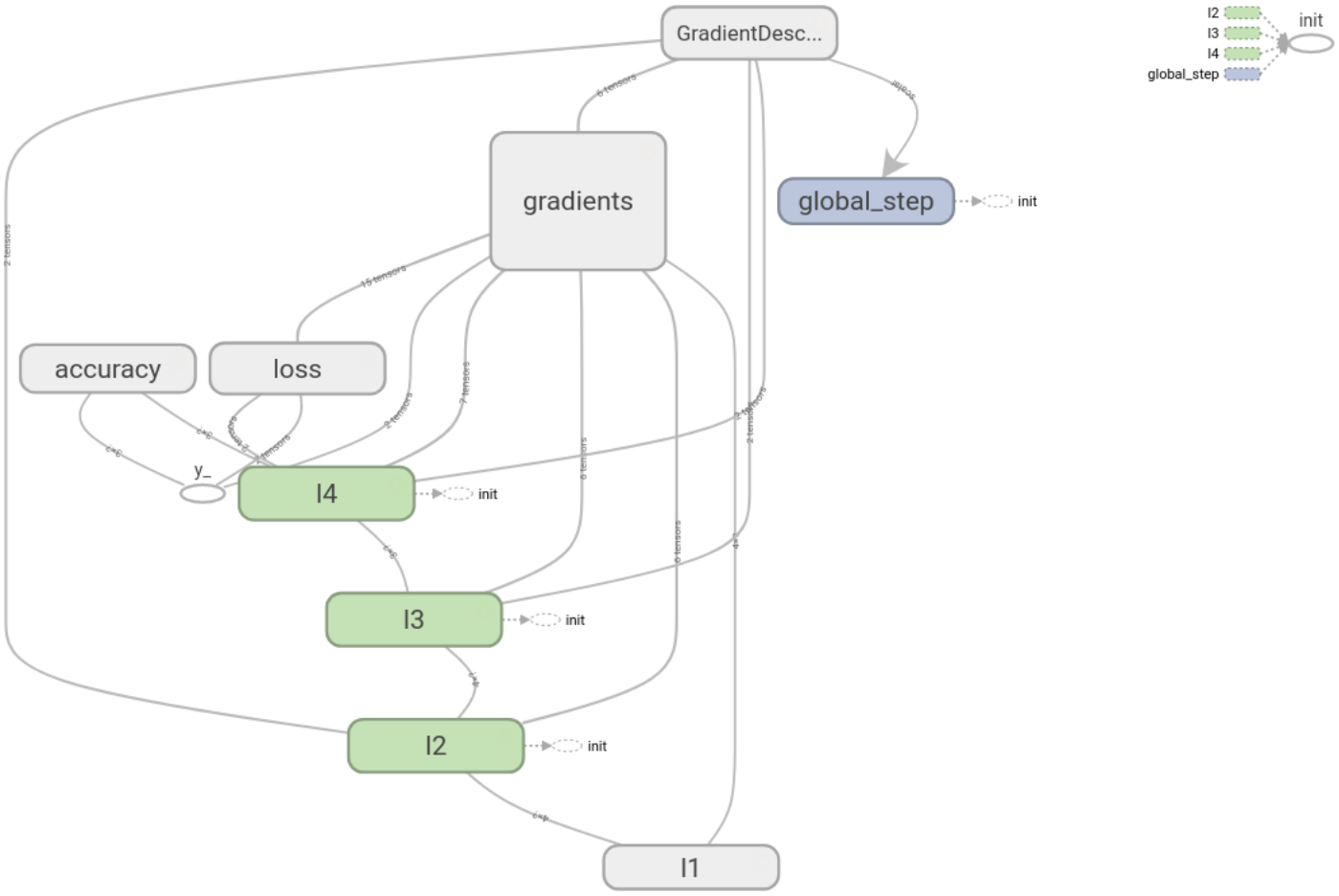
El dataset Iris



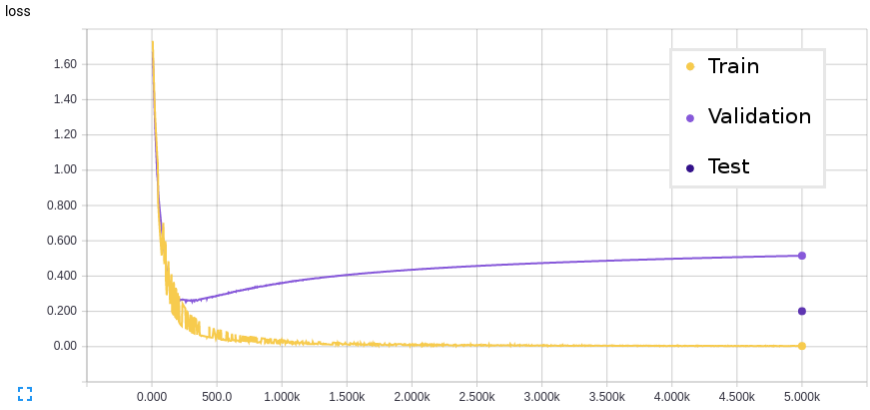
Crédito de imagen: *Kaggle*

- 150 muestras de flor = 50 $\times$ 3 subespecies: *Iris Setosa*, *Iris Versicolor* e *Iris Virginica*
- 4 datos de cada muestra: anchura y longitud de pétalo y sépalo
- Red elegida
  - **Activación**: $\sigma(x) = \frac{1}{1+e^{-x}}$
  - **Unidades**: 4, **4**, **3**, 3
  - **Coste**: *cross-entropy*
  - **Regularización**: L2

Perceptrón multicapa para Iris

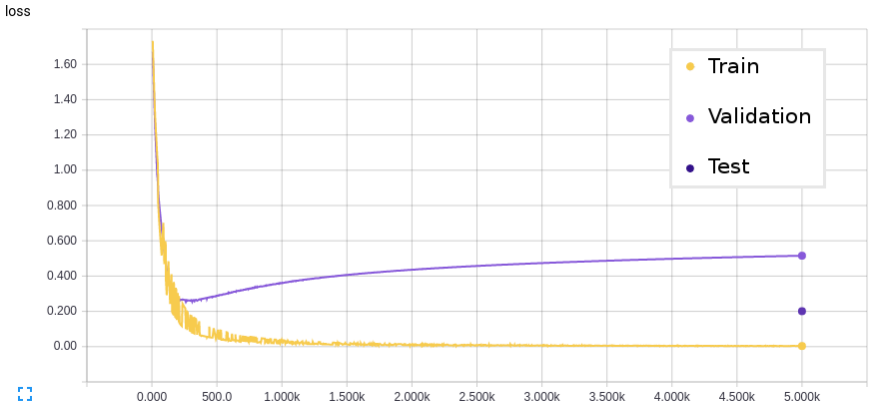


Regularización y *early stopping*

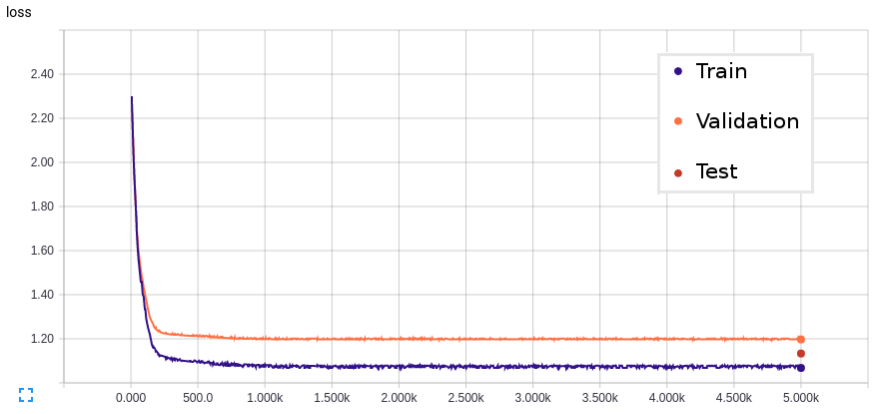


Sin regularización

Regularización y *early stopping*

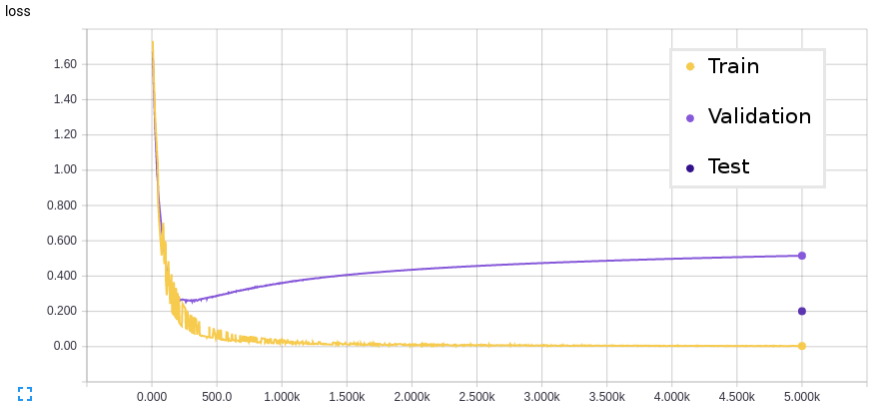


Sin regularización

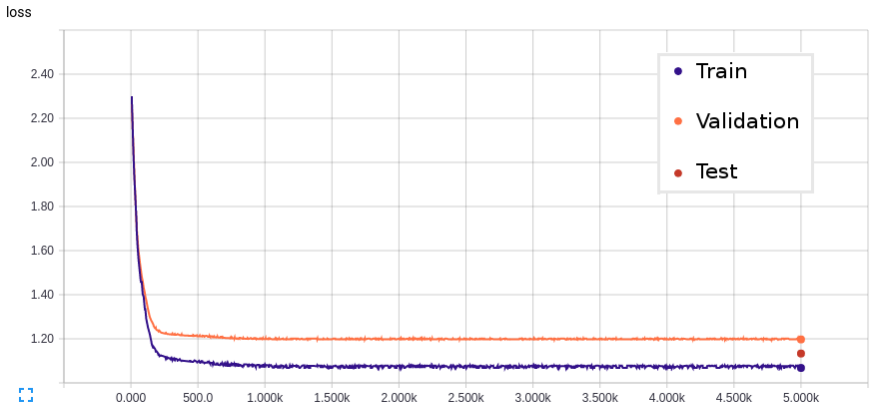


Regularización L2, $\lambda = 0{,}01$

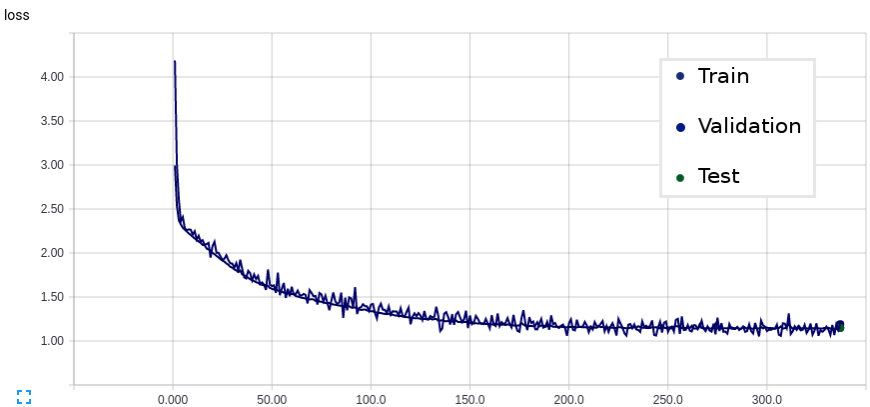
Regularización y *early stopping*



Sin regularización

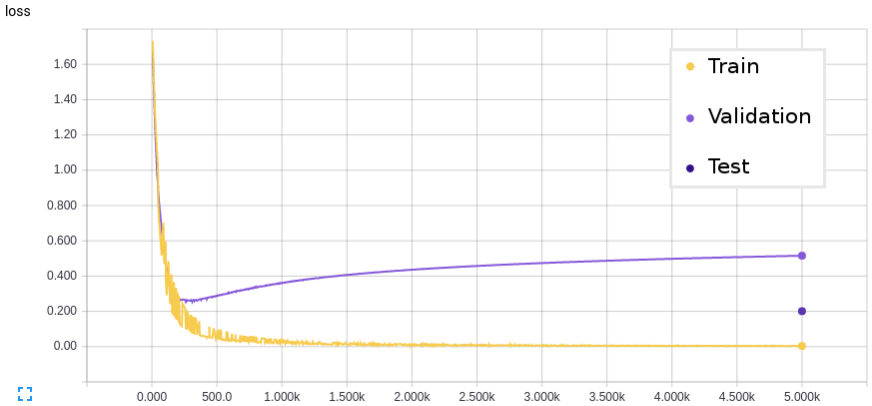


Regularización L2, $\lambda = 0{,}01$

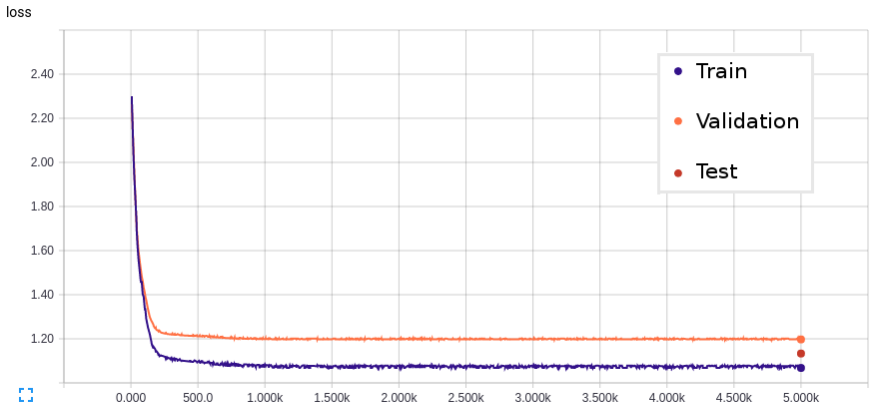


Early stopping (50 iteraciones)

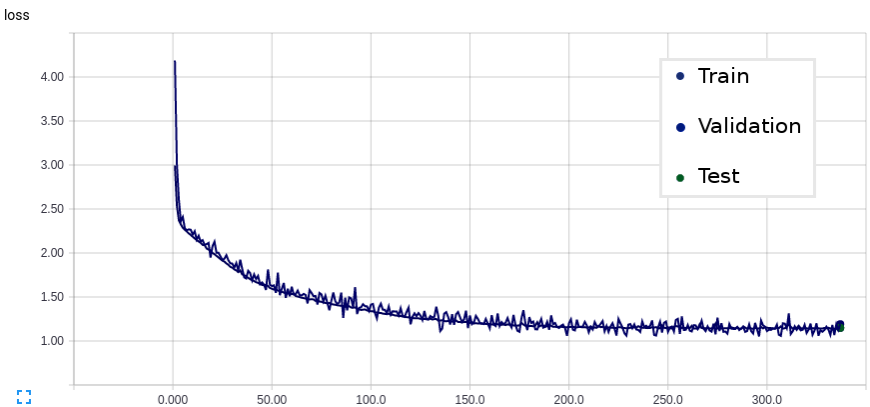
Regularización y *early stopping*



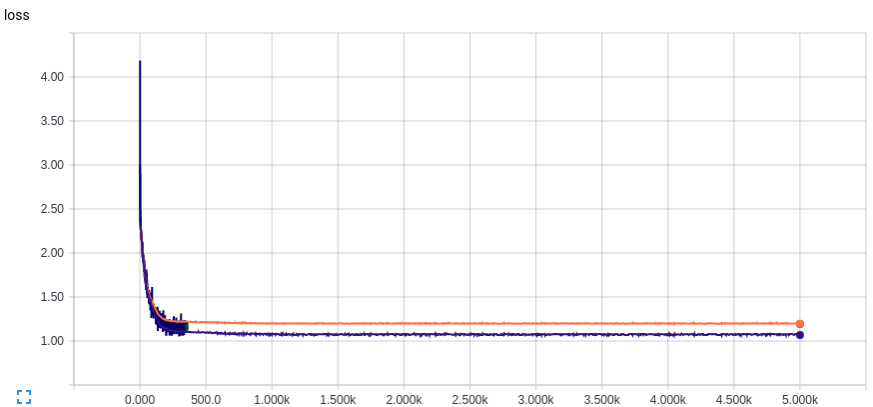
Sin regularización



Regularización L2, $\lambda = 0{,}01$



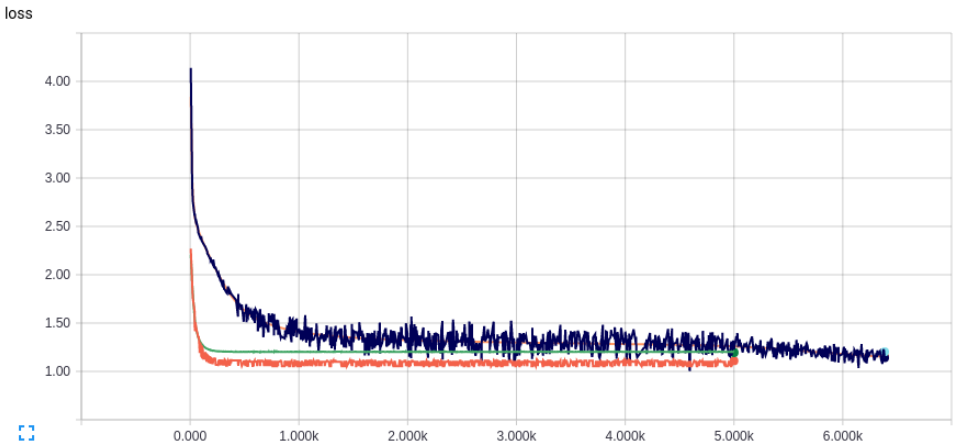
Early stopping (50 iteraciones)



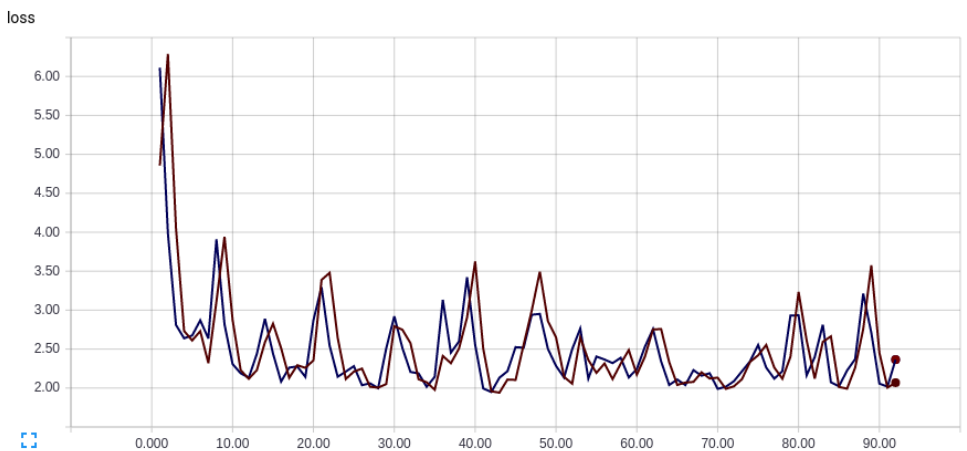
Comparación con/sin *early stopping*

Tasa de aprendizaje

α original: 1,0



Tasa de aprendizaje baja, $\alpha = 0{,}1$

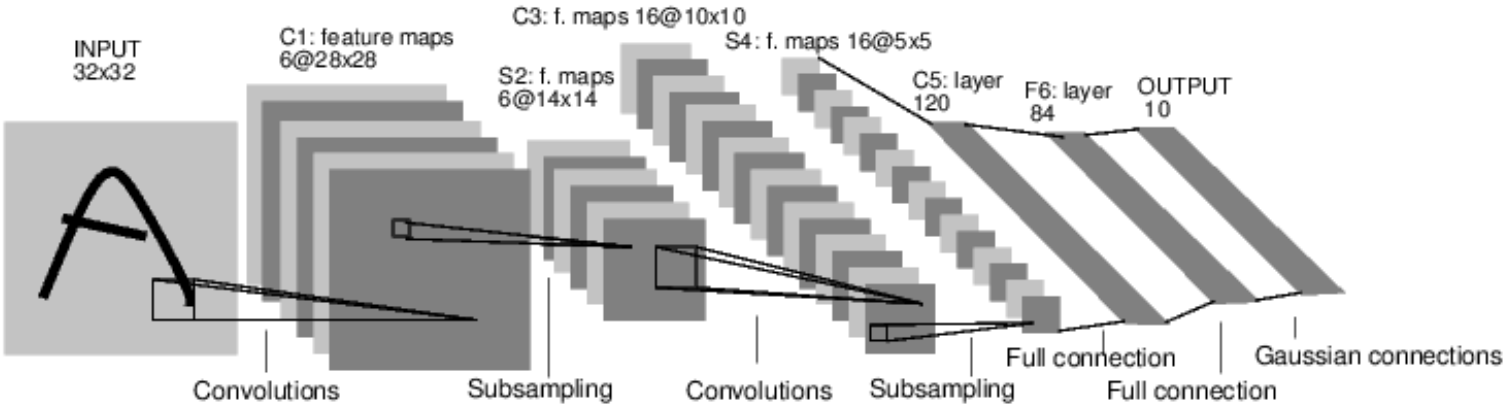


Tasa de aprendizaje alta, $\alpha = 10$

El dataset MNIST

- Reunido y curado por **Yann LeCun** en 1998
- **Train** + **Val**: 60k imágenes (55k + 5k).
  Test: 10k.
- Imágenes: 28×28, escala de grises
- Record actual: 0,21 % error en test

Fundamentos de ConvNets



LeNet-5. Crédito de la imagen: Yann LeCun

Preparación para MNIST

Ajuste de hiperparámetros

Entrenamiento (parámetros)

Feed-forward de input

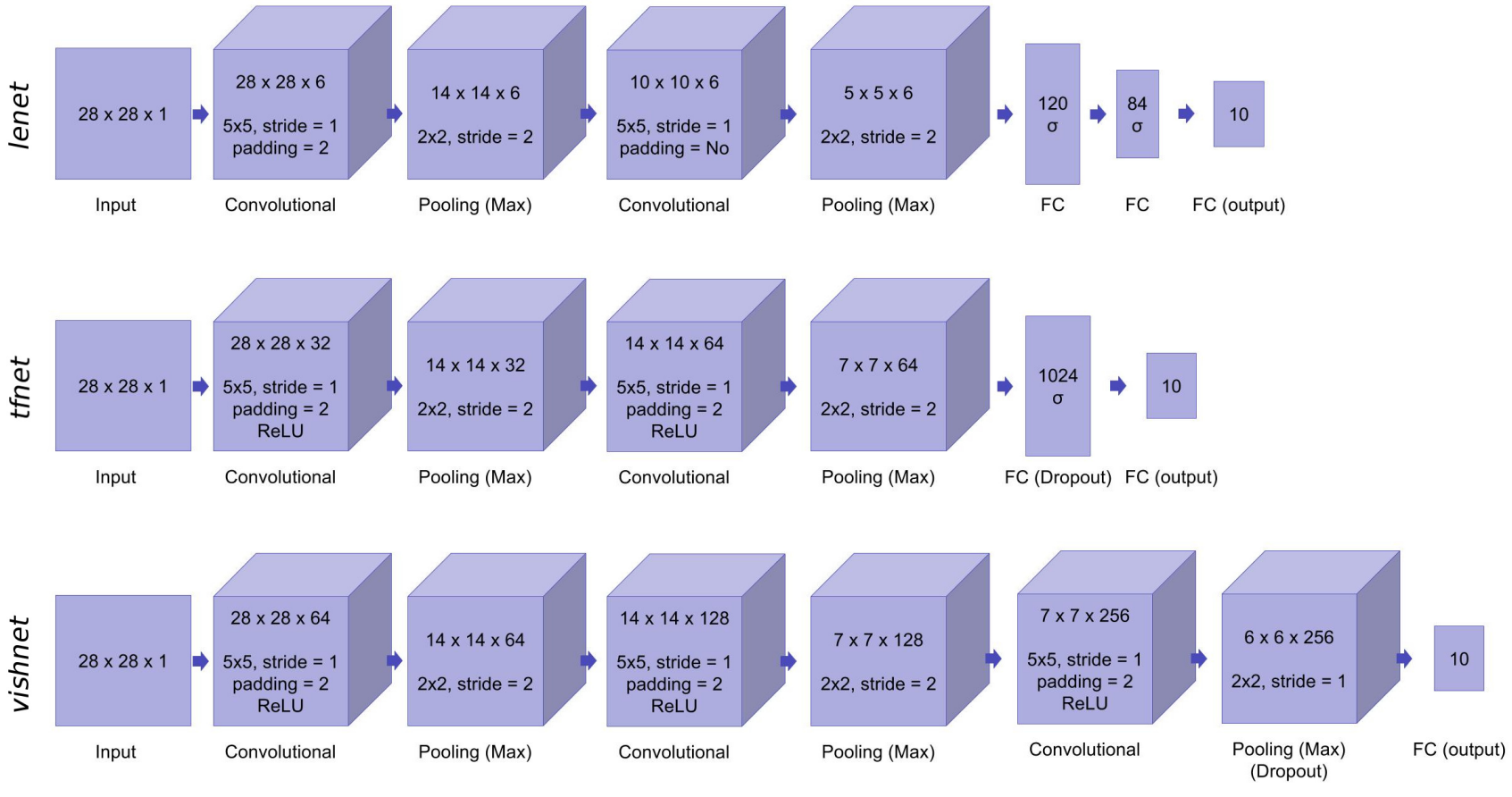
TensorFlow Slim

- **Capas** (convoluciones, pooling)
- **Costes** (cross-entropy y L2)
- ***ArgScopes***

Arquitecturas empleadas

1. `architectures.py`
   - Tres funciones que insertan cada una de las redes en el grafo:
     lenet: 61 470 pesos
     tfnet: 3 273 504 pesos
     vishnet: 1 117 760 pesos

Arquitecturas empleadas



Arquitecturas empleadas

1. `architectures.py`
   - Tres funciones que insertan cada una de las redes en el grafo:
     lenet: 61 470 pesos
     tfnet: 3 273 504 pesos
     vishnet: 1 117 760 pesos

2. `tuner.py`
   - Carcasa para `architectures.py`
   - Elementos comunes: cross-entropy, regularización L2, mini-batches de 50 elementos, etc.
   - *Grid search* sobre λ (regularización L2) y ρ (dropout)
   - Ejecutado en el **CCC UAM**

Resultados

- Valores hallados para los hiperparámetros:
  lenet (42 redes entrenadas): $\lambda = 4 \cdot 10^{-4}$
  tfnet (25 redes entrenadas): $\lambda = 5 \cdot 10^{-5}, \rho = 0{,}75$
  vishnet (25 redes entrenadas): $\lambda = 3 \cdot 10^{-7}, \rho = 0{,}4$

- Resultados del entrenamiento:

	5 000	10 000	20 000	40 000	60 000
lenet (0,064)	97,82 %	98,32 %	98,80 %	98,85 %	**98,82 %**
tfnet (0,59)	98,92 %	99,27 %	99,27 %	99,22 %	**99,36 %**
vishnet (1,75)	99,16 %	99,33 %	99,45 %	99,48 %	**99,51 %**

- Coste temporal: 4 $\times$ 135 000 pasos $\times$ 1,75 s/paso $\approx$ **262h** (*vishnet*). Total: **360h**

Conclusiones sobre TensorFlow

- A **plantearse**:
  - Funcionamiento propio, tiempo de aprendizaje
  - *API poco estable*
  - *Documentación mejorable*
- A **favor**:
  - Potencia (versión GPU, distribuida...)
  - Portabilidad multiplataforma
  - Google + colaborativo = éxito

Gracias por su atención