Taller de Aprendizaje Automático

Entrenamiento de Redes Neuronales Profundas (Parte 1)

Instituto de Ingeniería Eléctrica
Facultad de Ingeniería

1. Desvanecimiento y explosión del gradiente
    - Estrategias de inicialización de parámetros
    - Funciones de activación
    - Preprocesamiento y normalización de datos. *Batch-normalization*
    - Gradient clipping

2. Insuficiencia de datos: transfer learning

Entrenamiento de redes profundas: dificultades mayores

- Desvanecimiento o explosión del gradiente
- Cantidad de datos insuficientes para el problema a abordar
- Tiempo de entrenamiento
- Sobre-ajuste

1 Desvanecimiento y explosión del gradiente
Estrategias de inicialización de parámetros
Funciones de activación
Preprocesamiento y normalización de datos. *Batch-normalization*
Gradient clipping

2 Insuficiencia de datos: transfer learning

Desvanecimiento y explosión del gradiente

- El algoritmo de backpropagation propaga los gradientes de las capas de salida hacia las capas de entrada, propagando el error hacia atrás.
- Una vez que se ha calculado el gradiente de la función de costo con respecto a todos los parámetros de la red, el gradiente se utiliza para actualizar los parámetros con un paso de descenso por el gradiente.

Desvanecimiento y explosión del gradiente (cont.)

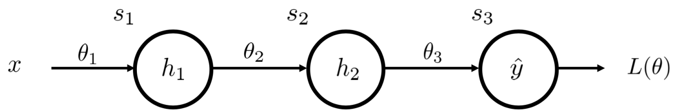
Problemas asociados a backpropagation en redes profundas (y en redes recurrentes):

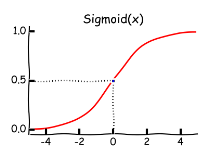
1. Propagación de errores.
2. Desvanecimiento del gradiente: es común que al propagarse hacia la entrada, los gradientes se vuelven cada vez más pequeños, afectando principalmente la actualización de los parámetros de la capa de entrada, que dejan de entrenarse.
3. Explosión del gradiente: puede suceder que los gradientes se amplifican alcanzando valores enormes en las primeras capas, haciendo que el algoritmo diverja (en especial en redes recurrentes).
4. Las capas pueden aprender a velocidades muy dispares.

Desvanecimiento y explosión del gradiente (cont.)

Ejemplo: red simple de tres capas,

$$\hat{y} = \sigma\left(\theta_3 \sigma\left(\theta_2 \sigma\left(\theta_1 x\right)\right)\right)$$





$$\sigma(s) = \frac{1}{1 + e^{-s}}$$

$$\sigma'(s) = \sigma(s)\left(1 - \sigma(s)\right)$$

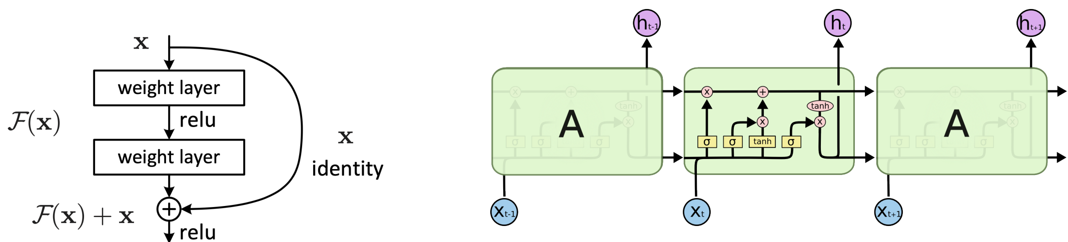
Gradiente con respecto a θ_3: $\dfrac{\partial L}{\partial \theta_3} = \dfrac{\partial L}{\partial \hat{y}} \dfrac{\partial \hat{y}}{\partial s_3} \dfrac{\partial s_3}{\partial \theta_3} = \dfrac{\partial L}{\partial \hat{y}} \sigma'\left(s_3\right) h_2$

Gradiente con respecto a θ_1: $\dfrac{\partial L}{\partial \theta_1} = \dfrac{\partial L}{\partial \hat{y}} \boxed{\sigma'\left(s_3\right)} h_2 \boxed{\sigma'\left(s_2\right)} h_1 \boxed{\sigma'\left(s_1\right)} x$

Multiplicación de valores $\sigma'(s_i) < 1 \Rightarrow$ atenuación exponencial.

Desvanecimiento y explosión del gradiente: soluciones

- Estrategias de inicialización de parámetros y funciones de activación*
- En redes convolucionales: redes residuales (ResNets)†
- En redes recurrentes: redes LSTM (Long Short-Term Memory)‡



*X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, pp. 249–256, May 2010

†K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016

‡S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997

1. Desvanecimiento y explosión del gradiente
   - Estrategias de inicialización de parámetros
   - Funciones de activación
   - Preprocesamiento y normalización de datos. *Batch-normalization*
   - Gradient clipping
2. Insuficiencia de datos: transfer learning

Inicialización de parámetros

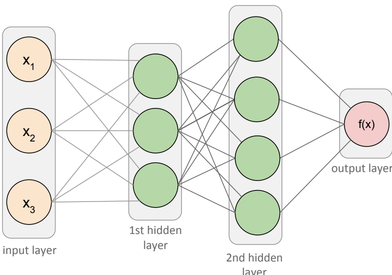
¿Qué sucede si inicializo los pesos $W = 0$? (u otra constante)

- **Problema de simetría:** todas las neuronas con las mismas conexiones producen la misma salida; descenso por gradiente actualiza todos los pesos de la misma manera

- Podríamos aplicar inicialización de pesos aleatoria:

$$W_{ij} \sim \mathcal{N}(0, \sigma^2), \qquad \sigma \text{ pequeño, e.g., } \sigma = 10^{-2}.$$

- Funciona bien en redes pequeñas.



Inicialización de parámetros

¿Qué sucede si inicializo $w_i \sim \mathcal{N}(0, \sigma^2)$, con e.g., $\sigma = 10^{-2}$?
$\mathrm{var}(a) = n \cdot \sigma^2 \cdot \mathrm{var}(x_1)$: Salida tiene distinta varianza que la entrada

$$
\begin{aligned}
\mathrm{var}(a) &= \mathrm{var}\left(\sum_{i=1}^{n} w_i x_i + b\right) \\
&= \sum_{i=1}^{n} \mathrm{var}(w_i x_i) \\
&= \sum_{i=1}^{n} E(x_i)^2 \mathrm{var}(w_i) + E(w_i)^2 \mathrm{var}(x_i) + \mathrm{var}(x_i)\mathrm{var}(w_i) \\
&= \sum_{i=1}^{n} \mathrm{var}(x_i)\mathrm{var}(w_i) \\
&= n \cdot \mathrm{var}(x_1)\mathrm{var}(w_1)
\end{aligned}
$$

Asumimos $E(x_i) = E(w_i) = 0$. (no siempre es verdadero, e.g., ReLu).

Inicialización de parámetros

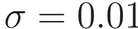
Ejemplo:

- Red totalmente conectada de 7 capas ocultas
- 1000 neuronas por capa
- 2000 datos de entrada
- Función de activación: $\tanh(x)$

```python
#Inicialización
nlayers=7
D = np.random.randn(2000,1000)
hidden_layer_sizes = [1000]*nlayers
```

```python
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i==0 else Hs[i-1]
    n_in = X.shape[1]
    n_out = hidden_layer_sizes[i]

    sigma = 0.01
    W = np.random.randn(n_in,n_out) * sigma

    H = np.dot(X,W)
    H = np.tanh(H)
    Hs[i] = H
```

Inicialización de parámetros



Layer 1 · Layer 2 · Layer 3 · Layer 4 · Layer 5 · Layer 6 · Layer 7

Layer means

Layer stds

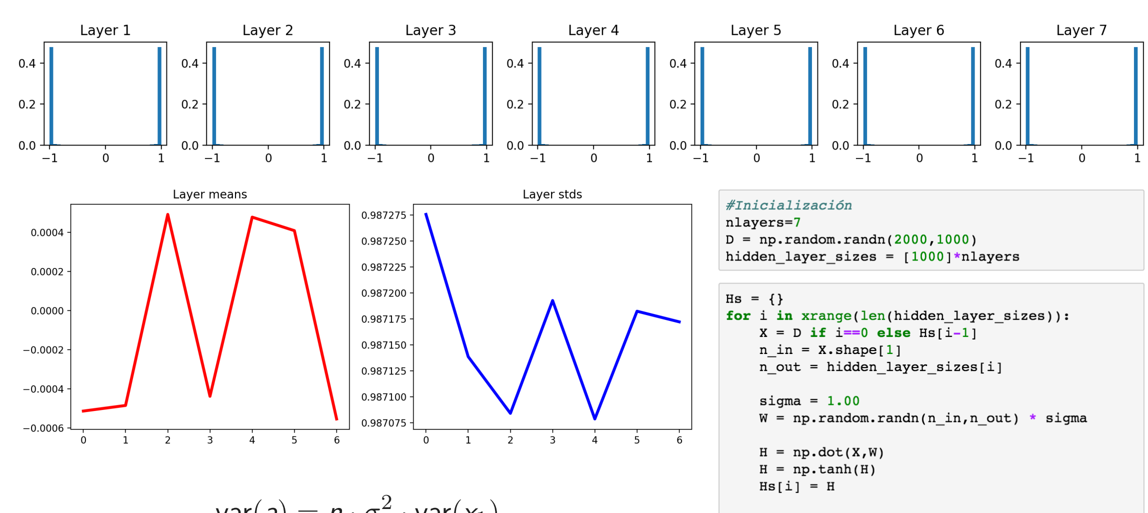
```
#Inicialización
nlayers=7
D = np.random.randn(2000,1000)
hidden_layer_sizes = [1000]*nlayers
```

```
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i==0 else Hs[i-1]
    n_in = X.shape[1]
    n_out = hidden_layer_sizes[i]

    sigma = 0.01
    W = np.random.randn(n_in,n_out) * sigma

    H = np.dot(X,W)
    H = np.tanh(H)
    Hs[i] = H
```

$$\mathrm{var}(a) = n \cdot \sigma^2 \cdot \mathrm{var}(x_1)$$

Inicialización de parámetros

$\sigma = 1.00$

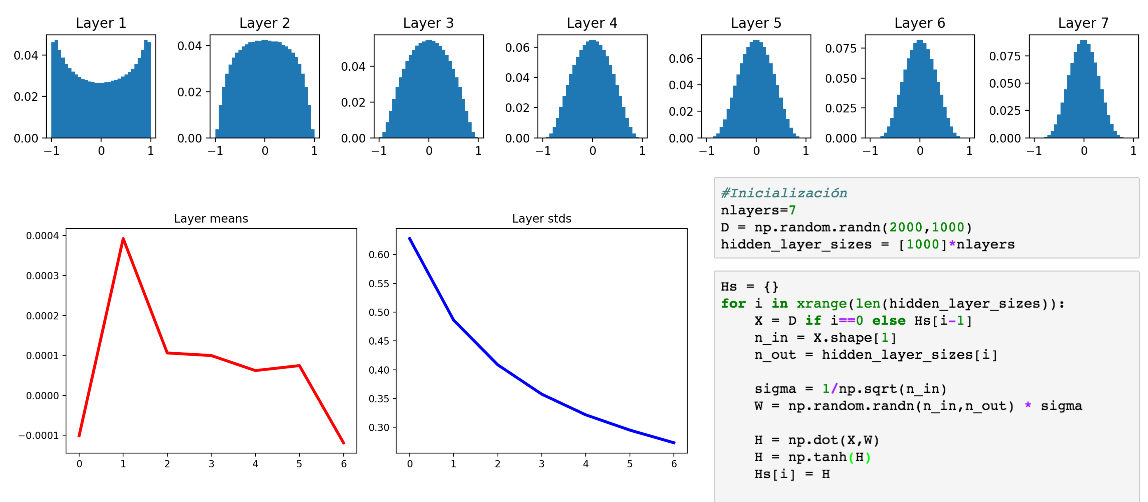



```
#Inicialización
nlayers=7
D = np.random.randn(2000,1000)
hidden_layer_sizes = [1000]*nlayers
```

```
Hs = {}
for i in xrange(len(hidden_layer_sizes)):
    X = D if i==0 else Hs[i-1]
    n_in = X.shape[1]
    n_out = hidden_layer_sizes[i]

    sigma = 1.00
    W = np.random.randn(n_in,n_out) * sigma

    H = np.dot(X,W)
    H = np.tanh(H)
    Hs[i] = H
```

$$\text{var}(a) = n \cdot \sigma^2 \cdot \text{var}(x_1)$$

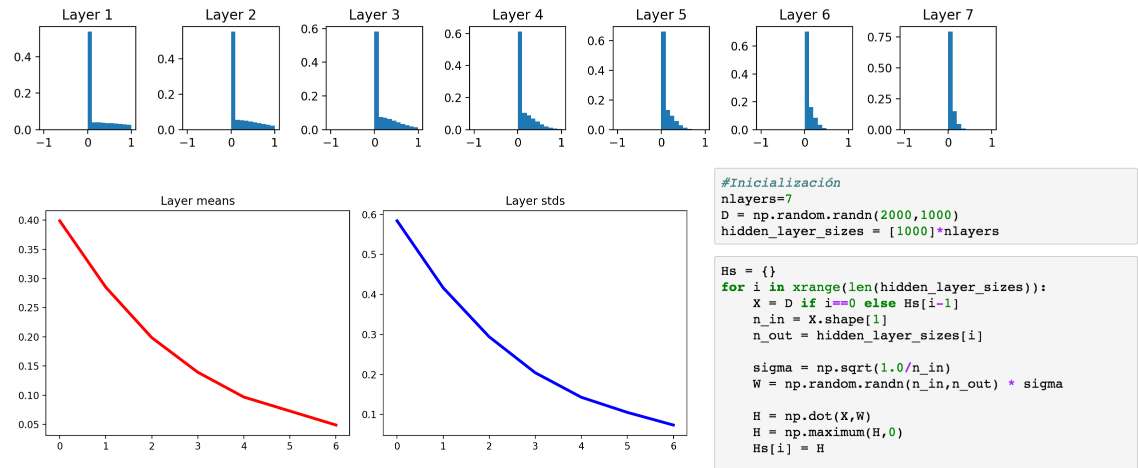
Inicialización de Xavier Glorot*: $\sigma_{\textbf{xavier}} = 1/\sqrt{n}$



*X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, pp. 249–256, May 2010

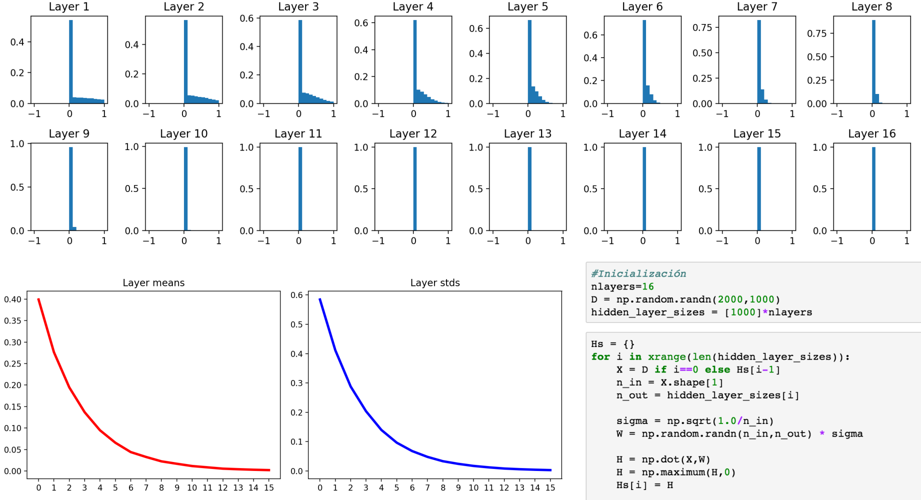
Inicialización de Xavier Glorot: $\sigma_{\textbf{xavier}} = 1/\sqrt{n}$

Con **ReLu:** desempeño pobre

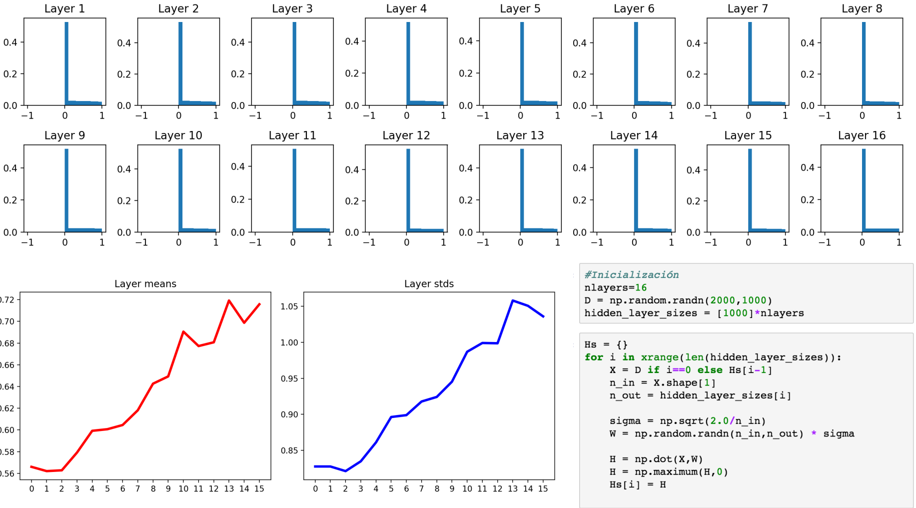


Inicialización de Xavier Glorot: $\sigma_{\textbf{xavier}} = 1/\sqrt{n}$

Con **ReLu:** desempeño pobre



Inicialización de Kaiming He*: $\sigma_{\textbf{kaiming}} = \sqrt{2/n}$ (ReLu)



*K. He, X. Zhang, S. Ren, and J. Sun, “Delving deep into rectifiers: Surpassing human-level performance on imagenet classification,” in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015

Inicialización de parámetros: área activa de investigación

- X. Glorot and Y. Bengio. *"Understanding the difficulty of training deep feedforward neural networks." ICAIS 2010.*
- I. Sutskever, J. Martens, G. Dahl, and G. Hinton. *"On the importance of initialization and momentum in deep learning."* ICML 2013.
- A. M. Saxe, J. L. McClelland and S. Ganguli. *"Exact solutions to the nonlinear dynamics of learning in deep linear neural networks."* ICLR 2014.
- K. He, X. Zhang, S. Ren, and J. Sun. *"Delving deep into rectifiers: Surpassing human-level performance on imagenet classification."* CVPR 2015.
- D. Mishkin and J. Matas.*"All you need is a good init".* ICLR 2016.
- Zhang et al. *"Fixup Initialization: Residual Learning Without Normalization".* ICLR 2019.
- J. Frankle and M. Carbin. *"The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks".* , ICLR 2019.

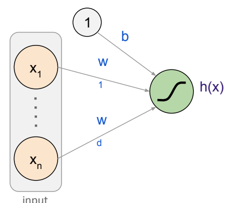
1 Desvanecimiento y explosión del gradiente
Estrategias de inicialización de parámetros
Funciones de activación
Preprocesamiento y normalización de datos. *Batch-normalization*
Gradient clipping

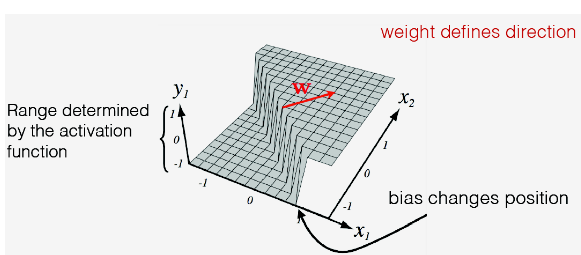
2 Insuficiencia de datos: transfer learning

Neuronas *artificiales*

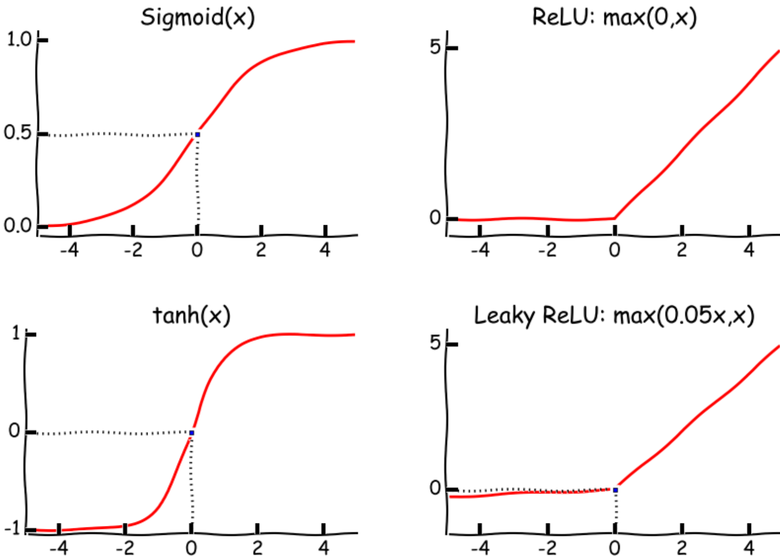
- Una neurona artificial se compone de: una operación lineal + una función de activación no lineal

$$h(\mathbf{x}) = g(a(\mathbf{x})) = g\left(\sum_{i=1}^{n} w_i x_i + b\right)$$

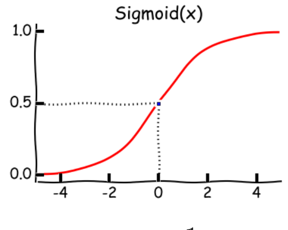




Funciones de Activación



Funciones de Activación



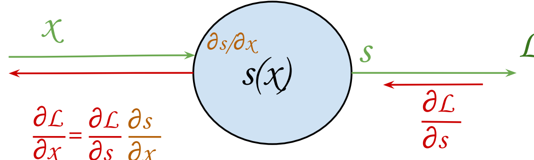
$$s(x) = \frac{1}{1 + e^{-x}}$$

$$s'(x) = s(x)(1 - s(x))$$

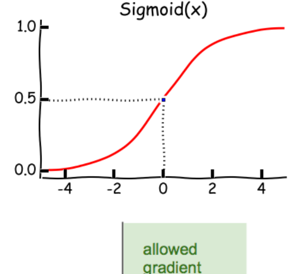
¿Qué pasa con el gradiente en zona saturada?

- $\frac{\partial s}{\partial x} \approx 0$: Muy difícil escapar de zona saturada
- Gradiente saliente = gradiente “local” multiplicado por gradiente entrante
- $\frac{\partial L}{\partial x} \approx 0$: El gradiente "hacia abajo" es cero

Los pesos de la red hacia abajo no se modifican
(por lo menos mediante este ramal)



Funciones de Activación



Salida es siempre positiva. ¿Es un problema?

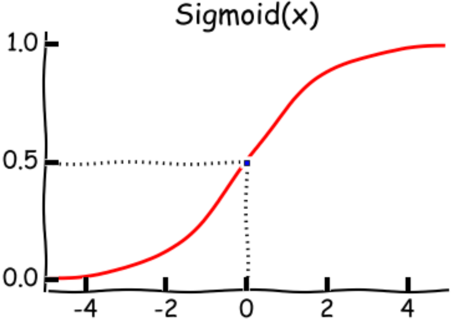
$$y = \sum_i w_i x_i + b \qquad \frac{\partial L}{\partial w_i} = \frac{\partial L}{\partial y} \underbrace{\frac{\partial y}{\partial w_i}}_{=x_i} = \frac{\partial L}{\partial y} x_i$$

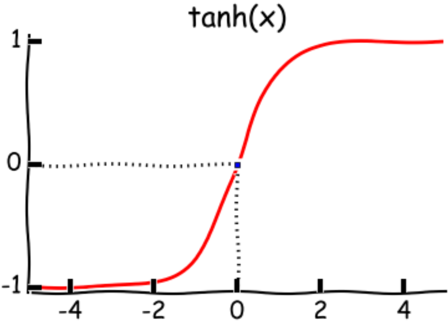
si $x_i \geq 0$, para todo i, entonces $\frac{\partial L}{\partial w_i}$ siempre tienen el mismo signo.

→ **Efecto zig-zag**

Es bueno tener datos/neuronas con media nula.

Funciones de Activación

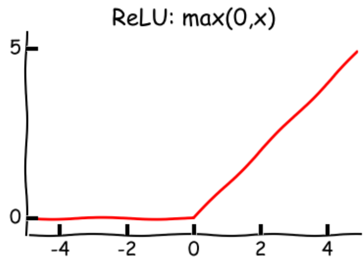




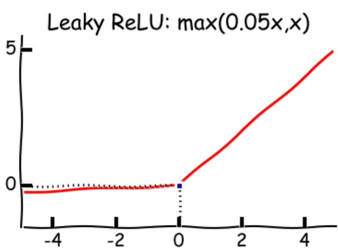
- Satura
- Salida siempre positiva.
- $\text{sigmoid}(x) = \frac{e^x}{1+e^x}$

- Satura
- **Salida en** $[-1, 1]$.
- $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$

Funciones de Activación



- No satura en $x \geq 0$.
- Salida es no negativa.
- Muy eficiente de calcular;
  muy utilizado en la práctica.
- **Neuronas Muertas**
  Si $x < 0$, $\frac{\partial s}{\partial x} = 0$: la neurona no puede cambiar de estado.
- En general inicializadas con **bias** levemente positivo (10^{-2}).

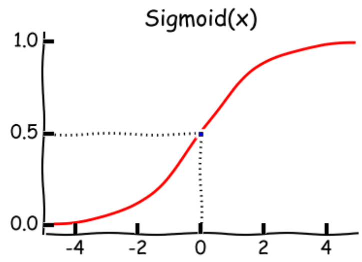


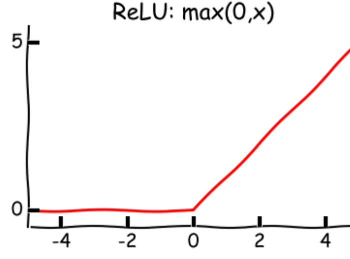
- Hereda todo lo bueno de ReLu
- y NO tiene el problema de neuronas muertas
- **Parametric Rectifier (PReLu):**

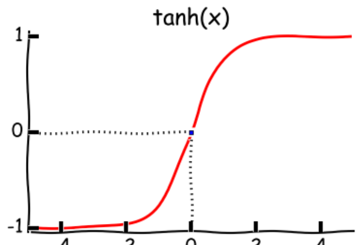
$$f(x) = \max(\alpha x, x)$$

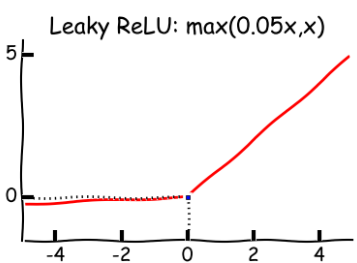
Parámetro α puede ser aprendido por *backprop*. [He. et al. 2015]

Funciones de Activación

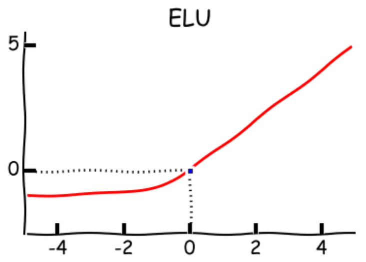


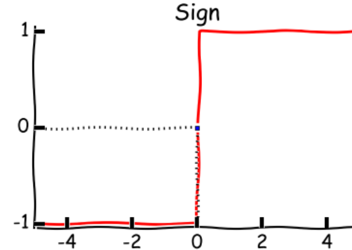


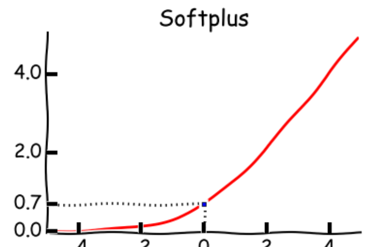


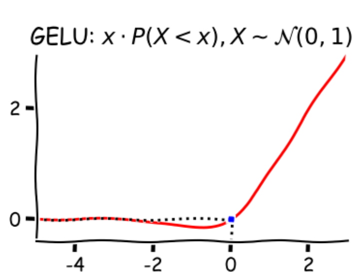


Funciones de Activación





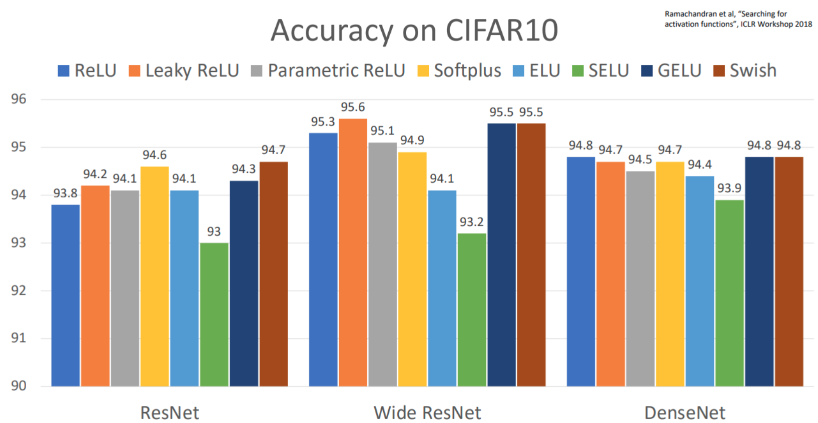




Funciones de Activación

Conclusión:

- Empezar probando con ReLU o LeakyReLU
- Otras variantes en general dan un incremento marginal (a veces necesario)
- No usar Sigmoide o Tanh en capas intermedias



Inicialización de parámetros y funciones de activación en Keras

Table 11-1. Initialization parameters for each type of activation function

Initialization	Activation functions	σ^2 (Normal)
Glorot	None, Tanh, Logistic, Softmax	$1 / fan_{avg}$
He	ReLU & variants	$2 / fan_{in}$
LeCun	SELU	$1 / fan_{in}$

```
keras.layers.Dense(10, activation="relu", kernel_initializer="he_normal")
```

```
he_avg_init = keras.initializers.VarianceScaling(scale=2., mode='fan_avg',
                                                 distribution='uniform')
keras.layers.Dense(10, activation="sigmoid", kernel_initializer=he_avg_init)
```

Inicialización de parámetros y funciones de activación en Keras

```
model = keras.models.Sequential([
    [...]
    keras.layers.Dense(10, kernel_initializer="he_normal"),
    keras.layers.LeakyReLU(alpha=0.2),
    [...]
])
```

```
layer = keras.layers.Dense(10, activation="selu",
                           kernel_initializer="lecun_normal")
```

1 Desvanecimiento y explosión del gradiente
Estrategias de inicialización de parámetros
Funciones de activación
Preprocesamiento y normalización de datos. *Batch-normalization*
Gradient clipping

2 Insuficiencia de datos: transfer learning

Preprocesamiento de datos

Queremos que datos tengan **media nula** :

- (i) para evitar fenómeno de **zig-zag** durante el aprendizaje;
- (ii) funciones tipo sigmoide/tanh operan en zona de no-saturacion;
- (iii) función de costo es menos sensible a valores de los parámetros.

En el caso de imágenes, podemos:

- Restar la imagen media (e.g. AlexNet)
- Restar la media de cada canal (e.g. VGG)
- Normalizar la varianza de cada canal (e.g. ResNet)
- Hacer otro pre-procesamiento ("blanqueo" tipo PCA). No es muy común en imágenes

Las estadísticas para hacer el pre-procesamiento tienen que ser aprendidas a partir del conjunto de entrenamiento

Batch Normalization[*]

Idea: Si lo que se quiere es datos con media nula y varianza fija ¿por qué no lo forzamos?

Sea $\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n \in \mathbb{R}^d$ un mini-batch de n datos (salida de una capa).

BatchNorm: Forzar que el minibatch tenga media 0 y varianza 1 (elemento a elemento).

Batch Normalization

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

$$\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{x}_i$$

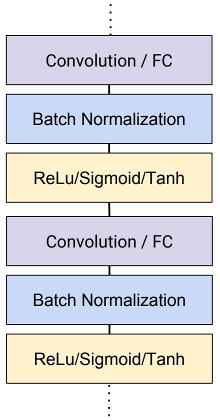
$$\boldsymbol{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \boldsymbol{\mu})^2}$$

Nota: Las operaciones $+, -, *, /$ son elemento a elemento, $\boldsymbol{\mu}, \boldsymbol{\sigma} \in \mathbb{R}^d$.

[*] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 448–456, JMLR.org, 2015

Batch Normalization

Sea $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n \in \mathbb{R}^d$ un mini-batch de n datos (salida de una capa)



Batch Normalization

$$\hat{\mathbf{x}}_i = \frac{\mathbf{x}_i - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

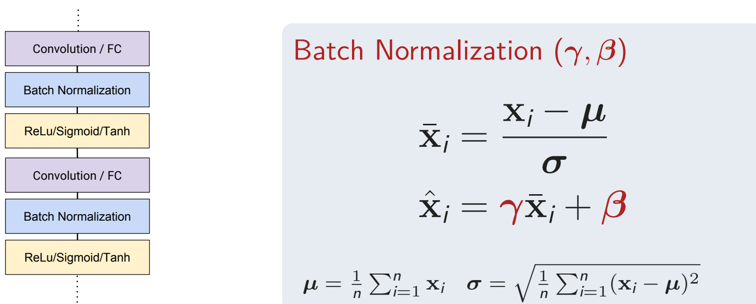
$\boldsymbol{\mu} = \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i$

$\boldsymbol{\sigma} = \sqrt{\frac{1}{n} \sum_{i=1}^{n} (\mathbf{x}_i - \boldsymbol{\mu})^2}$

Batch Normalization

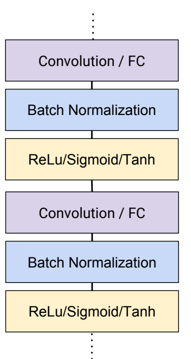
¿Es siempre mejor tener media nula y varianza uno?

- Forzar la salida de una capa a media nula, varianza uno puede limitar el poder de expresividad de la red (poca capacidad)
- Se agregan dos parámetros: γ, $\beta \in \mathbb{R}^d$



γ y β se aprenden mediante Backpropagation (BN: función diferenciable)

Batch Normalization



Batch Normalization (γ, β)

$$\bar{\mathbf{x}}_i = \frac{\mathbf{x}_i - \boldsymbol{\mu}}{\boldsymbol{\sigma}}$$

$$\hat{\mathbf{x}}_i = \gamma \bar{\mathbf{x}}_i + \beta$$

$\boldsymbol{\mu} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{x}_i \quad \boldsymbol{\sigma} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\mathbf{x}_i - \boldsymbol{\mu})^2}$

Observación. Podría llegar a suceder que: $\beta = \boldsymbol{\mu}$, $\gamma = \boldsymbol{\sigma}$ Se vuelve a obtener la estadística original (mapeo identidad).

Por un lado **BN** agrega más capacidad, pero además cambia la dinámica del entrenamiento haciéndola más fácil de entrenar.

Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \text{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

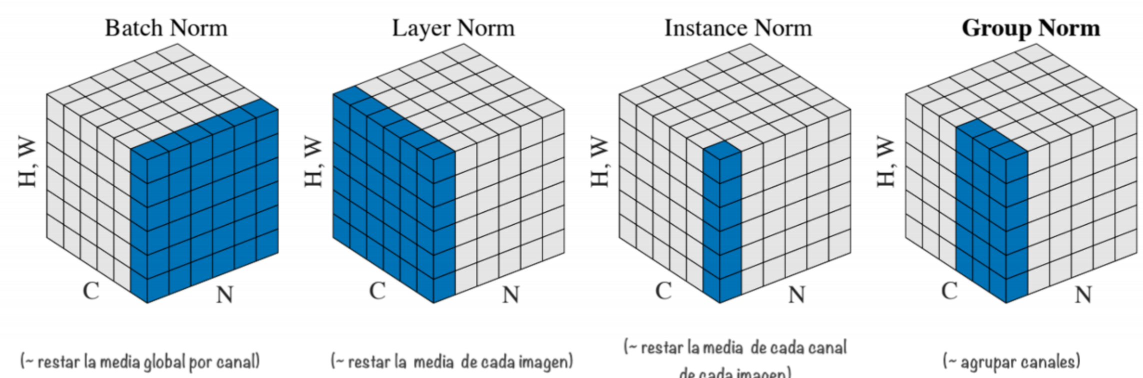
- Mejora robustez a inicialización.
- Mejora propagación del gradiente.
- Se comporta como un regularizador (agrega aleatoriedad).
- $\boldsymbol{\mu}$ y $\boldsymbol{\sigma}$ se calculan con los valores del mini batch actual.
- Permite acelerar el entrenamiento aumentando el *learning rate*

En *Test*:

Para aplicar BN en *test*, se almacena la **media** ($\boldsymbol{\mu}$) y **varianza** ($\boldsymbol{\sigma}^2$) de los datos durante el entrenamiento.

1. Dado un nuevo dato se corrige media y varianza usando valores **estimados** durante entrenamiento;
2. Se re-escala y se traslada por γ y β respectivamente

Batch Normalization en capas convolucionales



Wu and He, "Group Normalization", ECCV 2018

Batch Normalization en Keras

```python
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(300, activation="elu",
kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(100, activation="elu",
kernel_initializer="he_normal"),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(10, activation="softmax")
])
```

```
>>> model.summary()
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
flatten_3 (Flatten)          (None, 784)               0
_________________________________________________________________
batch_normalization_v2 (Batc (None, 784)               3136
_________________________________________________________________
dense_50 (Dense)             (None, 300)               235500
_________________________________________________________________
batch_normalization_v2_1 (Ba (None, 300)               1200
_________________________________________________________________
dense_51 (Dense)             (None, 100)               30100
_________________________________________________________________
batch_normalization_v2_2 (Ba (None, 100)               400
_________________________________________________________________
dense_52 (Dense)             (None, 10)                1010
=================================================================
Total params: 271,346
Trainable params: 268,978
Non-trainable params: 2,368
```

- La capa `BatchNormalization` agrega 4 parámetros $(\gamma, \beta, \mu, \sigma)$ por entrada.
- Ejemplo: la primera capa de `BatchNormalization` introduce $4 \times 784 = 3136$ parámetros.
- Keras califica a los parámetros μ y σ como "no-entrenables" (son aprenden de los datos pero no son modificados por backpropagation).
- Esto da un total de $(3136 + 1200 + 400)/2 = 2368$ parámetros no-entrenables.

Batch Normalization en Keras

Inspección de los parámetros de la primera capa `BatchNormalization`:

```
>>> [(var.name, var.trainable) for var in model.layers[1].variables]
[('batch_normalization_v2/gamma:0', True),
 ('batch_normalization_v2/beta:0', True),
 ('batch_normalization_v2/moving_mean:0', False),
 ('batch_normalization_v2/moving_variance:0', False)]
```

```
>>> model.layers[1].updates
[<tf.Operation 'cond_2/Identity' type=Identity>,
 <tf.Operation 'cond_3/Identity' type=Identity>]
```

Keras crea para la capa dos operaciones de Tensorflow que serán las encargadas de actualizar las medias y varianzas móviles.

Batch Normalization en Keras

Si bien es debatido, los autores de Batch Normalization consideran que es mejor incluir estas capas antes de las funciones de activación. Esto se implementa así:

```
model = keras.models.Sequential([
    keras.layers.Flatten(input_shape=[28, 28]),
    keras.layers.BatchNormalization(),
    keras.layers.Dense(300, kernel_initializer="he_normal",
use_bias=False),
    keras.layers.BatchNormalization(),
    keras.layers.Activation("elu"),
    keras.layers.Dense(100, kernel_initializer="he_normal",
use_bias=False),
    keras.layers.BatchNormalization(),
    keras.layers.Activation("elu"),
    keras.layers.Dense(10, activation="softmax")
])
```

Batch Normalization en Keras

Hiperparámetros

- momentum: parámetro utilizado en la actualización de las medias móviles exponenciales. Dado un nuevo valor $\mathbf{v}$ (i.e. un nuevo vector de medias o desvíos estándar calculados sobre el batch actual), la capa actualiza la media móvil según

$$\hat{\mathbf{v}} \leftarrow \texttt{momentum} \times \hat{\mathbf{v}} + (1 - \texttt{momentum}) \times \mathbf{v}.$$

- axis: indica el eje o coordenada en la que se normaliza. El valor por defecto es -1 (última).
  Cuando el batch de entrada es 2D, de tamaño [*tamaño de batch, features*], cada *feature* es normalizada por su media y desvío estándar calculado sobre todas las muestras del batch.

1 Desvanecimiento y explosión del gradiente
Estrategias de inicialización de parámetros
Funciones de activación
Preprocesamiento y normalización de datos. *Batch-normalization*
Gradient clipping

2 Insuficiencia de datos: transfer learning

Gradient Clipping

- Permite mitigar el problema de la explosión de los gradientes.
- Consiste en hacer un clipping de las coordenadas del gradiente a un cierto valor cuando éstas lo exceden. En Keras:

```
optimizer = keras.optimizers.SGD(clipvalue=1.0)
model.compile(loss="mse", optimizer=optimizer)
```

- En lugar de hacer un clipping por coordenada, que alterará la dirección del gradiente, conviene hacer un clipping según la norma del gradiente usando `clipnorm` en lugar de `clipvalue`. Si la norma excede el valor de `clipvalue`, todo el vector se escala a norma ℓ_2 unitaria.
- En general se opta por *Batch Normalization* en lugar de *gradient clipping*, salvo en redes recurrentes en donde la implementación de *Batch Normalization* es compleja.

1. Desvanecimiento y explosión del gradiente
   - Estrategias de inicialización de parámetros
   - Funciones de activación
   - Preprocesamiento y normalización de datos. *Batch-normalization*
   - Gradient clipping

2. Insuficiencia de datos: transfer learning

Aprendizaje por Transferencia *(Transfer Learning)*

- Entrenar una red con millones de parámetros requiere contar con un conjunto de datos muy grande
- ¿Esto significa que *deep learning* no se puede utilizar si no tenemos muchos datos?
- **No!** Se puede utilizar mediante aprendizaje por transferencia:
  1. Se entrena en una base similar (e.g., si datos son imágenes, Imagenet)
  2. Se re-entrena la últimas capas de la red con el conjunto de datos específicos

Aprendizaje por Transferencia *(Transfer Learning)**

DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition

Jeff Donahue*, Yangqing Jia*, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, Trevor Darrell
{JDONAHUE,JIAYQ,VINYALS,JHOFFMAN,NZHANG,ETZENG,TREVOR}@EECS.BERKELEY.EDU
UC Berkeley & ICSI, Berkeley, CA, USA

Abstract

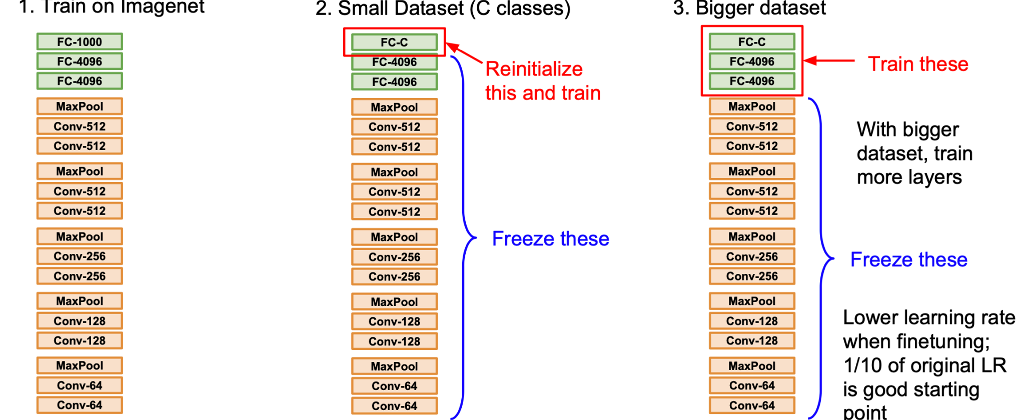
We evaluate whether features extracted from the activation of a deep convolutional network trained in a fully supervised fashion on a large, fixed set of object recognition tasks can be re-purposed to novel generic tasks. Our generic tasks may differ significantly from the originally trained tasks and there may be insufficient labeled or unlabeled data to conventionally train or adapt a deep architecture to the new tasks. We investigate and visualize the semantic clustering of deep convolutional features with respect to a variety of such tasks, including scene recognition, domain adaptation, and fine-grained recognition challenges. We compare the efficacy of relying on various network levels to define a fixed fea-

pects of a given domain through discovery of salient clusters, parts, mid-level features, and/or hidden units (Hinton & Salakhutdinov, 2006; Fidler & Leonardis, 2007; Zhu et al., 2007; Singh et al., 2012; Krizhevsky et al., 2012). Such models have been able to perform better than traditional hand-engineered representations in many domains, especially those where good features have not already been engineered (Le et al., 2011). Recent results have shown that moderately deep unsupervised models outperform the state-of-the art gradient histogram features in part-based detection models (Ren & Ramanan, 2013).

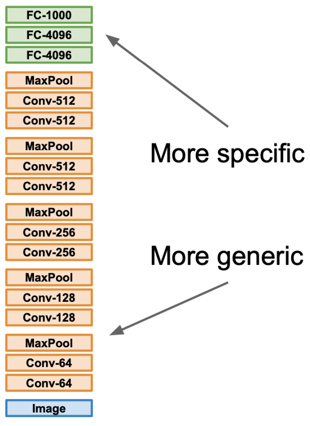
Deep models have recently been applied to large-scale visual recognition tasks, trained via back-propagation through layers of convolutional filters (LeCun et al., 1989). These models perform extremely well in domains with large amounts of training data, and had early success in

* J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, “Decaf: A deep convolutional activation feature for generic visual recognition,” in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML’14, p. I–647–I–655, JMLR.org, 2014

Aprendizaje por Transferencia *(Transfer Learning)*



Aprendizaje por Transferencia *(Transfer Learning)*



	very similar dataset	**very different dataset**
very little data	Use Linear Classifier on top layer	You're in trouble… Try linear classifier from different stages
quite a lot of data	Finetune a few layers	Finetune a larger number of layers

Aprendizaje por Transferencia *(Transfer Learning)*

Ejemplo*

Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields *

Zhe Cao Tomas Simon Shih-En Wei Yaser Sheikh
The Robotics Institute, Carnegie Mellon University
{zhecao,shihenw}@cmu.edu {tsimon,yaser}@cs.cmu.edu

Abstract

We present an approach to efficiently detect the 2D pose of multiple people in an image. The approach uses a non-parametric representation, which we refer to as Part Affinity Fields (PAFs), to learn to associate body parts with individuals in the image. The architecture encodes global context, allowing a greedy bottom-up parsing step that maintains high accuracy while achieving realtime performance, irrespective of the number of people in the image. The architecture is designed to jointly learn part locations and their association via two branches of the same sequential prediction process. Our method placed first in the inaugural COCO 2016 keypoints challenge, and significantly exceeds the previous state-of-the-art result on the MPII Multi-Person benchmark, both in performance and efficiency.

Figure 1. **Top:** Multi-person pose estimation. Body parts belonging to the same person are linked. **Bottom left:** Part Affinity Fields (PAFs) corresponding to the limb connecting right elbow and right wrist. The color encodes orientation. **Bottom right:** A zoomed in view of the predicted PAFs. At each pixel in the field, a 2D vector

*Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017

Aprendizaje por Transferencia *(Transfer Learning)*

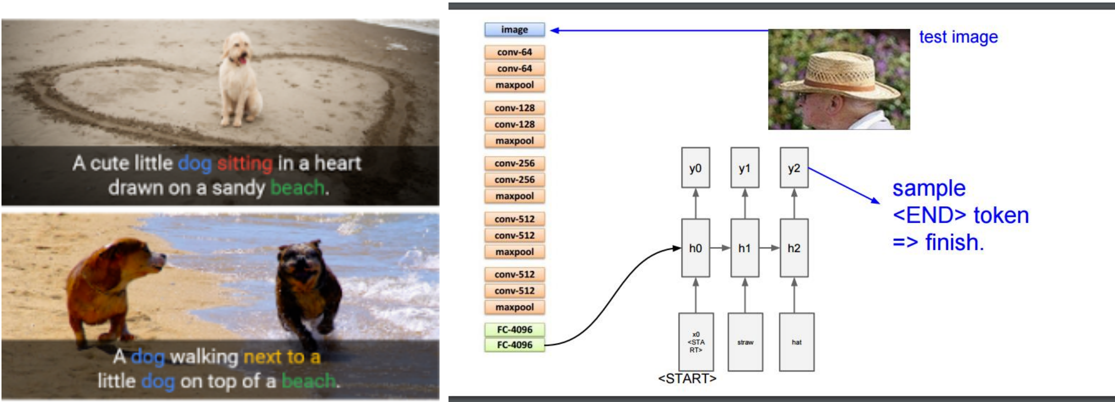
Ejemplo*

tion architecture, following Wei et al. [31], which refines the predictions over successive stages, $t \in \{1, \ldots, T\}$, with intermediate supervision at each stage.

The image is first analyzed by a convolutional network (initialized by the first 10 layers of VGG-19 [26] and fine-tuned), generating a set of feature maps $\mathbf{F}$ that is input to the first stage of each branch. At the first stage, the network produces a set of detection confidence maps $\mathbf{S}^1 = \rho^1(\mathbf{F})$ and a set of part affinity fields $\mathbf{L}^1 = \phi^1(\mathbf{F})$, where ρ^1 and ϕ^1 are the CNNs for inference at Stage 1. In each subsequent stage, the predictions from both branches in the previous stage, along with the original image features $\mathbf{F}$, are concatenated and used to produce refined predictions,

*Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017

Aprendizaje por Transferencia *(Transfer Learning)*



Slides adaptada de cs231n (Stanford) - Fei-Fei Li & Justin Johnson & Serena Yeung

Transfer Learning en Keras

Ejemplo:

- Supongamos que la base Fashion MNIST contiene 8 clases (las 10 originales salvo sandal y shirt.) Se entrena un modelo sobre esta base y se obtiene una performance razonable (más de 90% de accuracy). Lo llamamos modelo A.
- Ahora queremos diseñar un clasificador binario para las clases sandal y shirt, para lo cual disponemos de apenas 200 muestras etiquetadas.
- Para esto se entrena un nuevo modelo (modelo B), usando la misma arquitectura que el model A (a excepción de la última capa), con un desempeño de 97.2% de accuracy.
- Tratándose de un problema mucho más sencillo (dos clases), esperábamos un resultado mejor... intentemos mejorar el modelo B usando *transfer learning* a partir del modelo A.

Transfer Learning en Keras

1. Cargamos el modelo A
2. Creamos el modelo B a partir del modelo A salvo la última capa, y agregamos la capa totalmente conectada para el problema binario
3. Para no perder la versión del modelo A con los pesos ya entrenados, generamos un clon y copiamos sus pesos (al entrenar `model_B_on_A`, se modifican los pesos de `model_A`)
4. Para focalizar el entrenamiento en los pesos de la última capa en las primeras épocas, congelamos los pesos de las capas anteriores.
5. Compilamos `model_B_on_A` (siempre debe hacerse antes de congelar o descongelar capas).

```
model_A = keras.models.load_model("my_model_A.h5")
model_B_on_A = keras.models.Sequential(model_A.layers[:-1])
model_B_on_A.add(keras.layers.Dense(1, activation="sigmoid"))
model_A_clone = keras.models.clone_model(model_A)
model_A_clone.set_weights(model_A.get_weights())
```

```
for layer in model_B_on_A.layers[:-1]:
    layer.trainable = False

model_B_on_A.compile(loss="binary_crossentropy", optimizer="sgd",
                     metrics=["accuracy"])
```

Transfer Learning en Keras

1. Entrenamos `model_B_on_A` unas pocas épocas
2. Descongelamos los pesos de las capas copiados de `model_A`, para hacer un ajuste fino de toda la red
3. Para evitar perturbar demasiado los pesos de las capas copiadas de `model_A`, es conveniente reducir el learning rate (`lr`)
4. Volvemos a compilar `model_B_on_A`
5. Entrenamos toda la red a partir de los pesos actuales

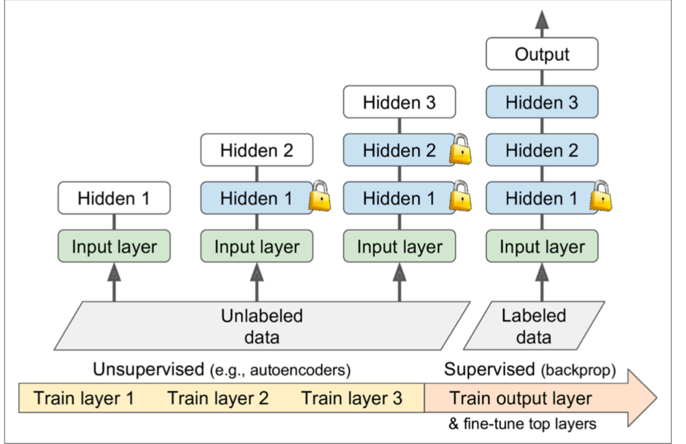
```
history = model_B_on_A.fit(X_train_B, y_train_B, epochs=4,
                           validation_data=(X_valid_B, y_valid_B))

for layer in model_B_on_A.layers[:-1]:
    layer.trainable = True

optimizer = keras.optimizers.SGD(lr=1e-4) # the default lr is 1e-2
model_B_on_A.compile(loss="binary_crossentropy", optimizer=optimizer,
                     metrics=["accuracy"])
history = model_B_on_A.fit(X_train_B, y_train_B, epochs=16,
                           validation_data=(X_valid_B, y_valid_B))
```

Con esta estrategia, refinando lo suficiente, es posible aumentar sensiblemente el desempeño.

```
>>> model_B_on_A.evaluate(X_test_B, y_test_B)
[0.06887910133600235, 0.9925]
```

Pre-entrenamiento no supervisado

Greedy layer-wise pretraining*



* G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, p. 1527–1554, July 2006

Referencias I

X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *JMLR W&CP: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, vol. 9, pp. 249–256, May 2010.

K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026–1034, 2015.

S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, p. 448–456, JMLR.org, 2015.

J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *Proceedings of the 31st International Conference on International Conference on Machine Learning - Volume 32*, ICML'14, p. I–647–I–655, JMLR.org, 2014.

Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, p. 1527–1554, July 2006.

A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 3rd Edition*.
O'Reilly Media, Inc., 2022.

T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*.
Springer Series in Statistics, New York, NY, USA: Springer New York Inc., 2001.

Referencias II

C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*.
Berlin, Heidelberg: Springer-Verlag, 2006.

Y. Nesterov, "A method for solving the convex programming problem with convergence rate $o(1/k^2)$," *Proceedings of the USSR Academy of Sciences*, vol. 269, pp. 543–547, 1983.

I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, p. III–1139–III–1147, JMLR.org, 2013.

J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, p. 2121–2159, July 2011.

D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.
cite arxiv:1207.0580.

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, p. 1929–1958, Jan. 2014.

Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *Proceedings of The 33rd International Conference on Machine Learning* (M. F. Balcan and K. Q. Weinberger, eds.), vol. 48 of *Proceedings of Machine Learning Research*, (New York, New York, USA), pp. 1050–1059, PMLR, 20–22 Jun 2016.