Aprendizaje profundo basado en la física

Semana 2: aprendizaje profundo y constraints

Docente: José I. Robledo - 14/04/2026

Repaso loop entrenamiento en PyTorch

for epoch in range(num_epochs):

    for x_batch, y_batch in dataloader:

        1) cargar datos batch

        2) resetear gradientes

        3) forward

        4) calcular loss

        5) backward

        6) update de parámetros

Época

└── Batch

    ├── x_batch, y_batch

    ├── optimizer.zero_grad()

    ├── y_hat = model(x_batch)

    ├── loss = criterion(y_hat, y_batch)

    ├── loss.backward()

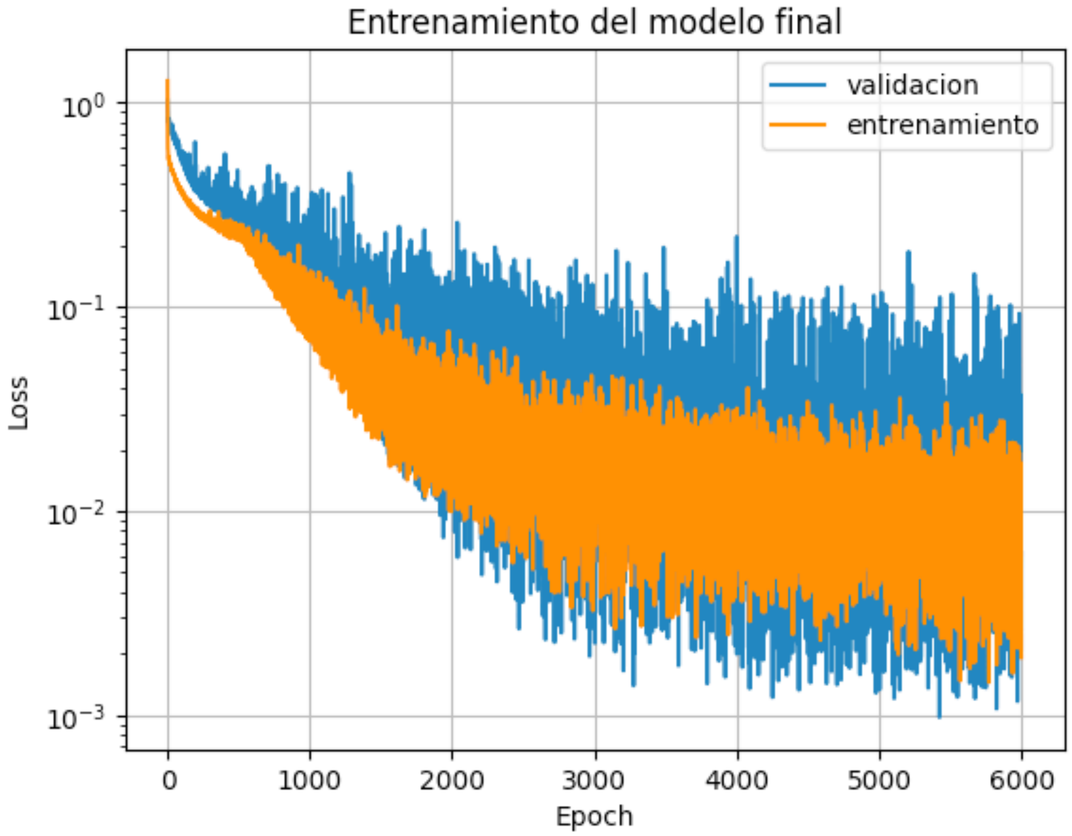
    └── optimizer.step()

Repaso loop entrenamiento en PyTorch

```
for epoch in range(num_epochs):
    model.train()
    for x_batch, y_batch in train_loader:
        optimizer.zero_grad()
        y_hat = model(x_batch)
        loss = criterion(y_hat, y_batch)
        loss.backward()
        optimizer.step()

    model.eval()
    with torch.no_grad():
        for x_val, y_val in val_loader:
            y_val_hat = model(x_val)
            val_loss = criterion(y_val_hat, y_val)
```

Train vs. eval



Cálculo del gradiente

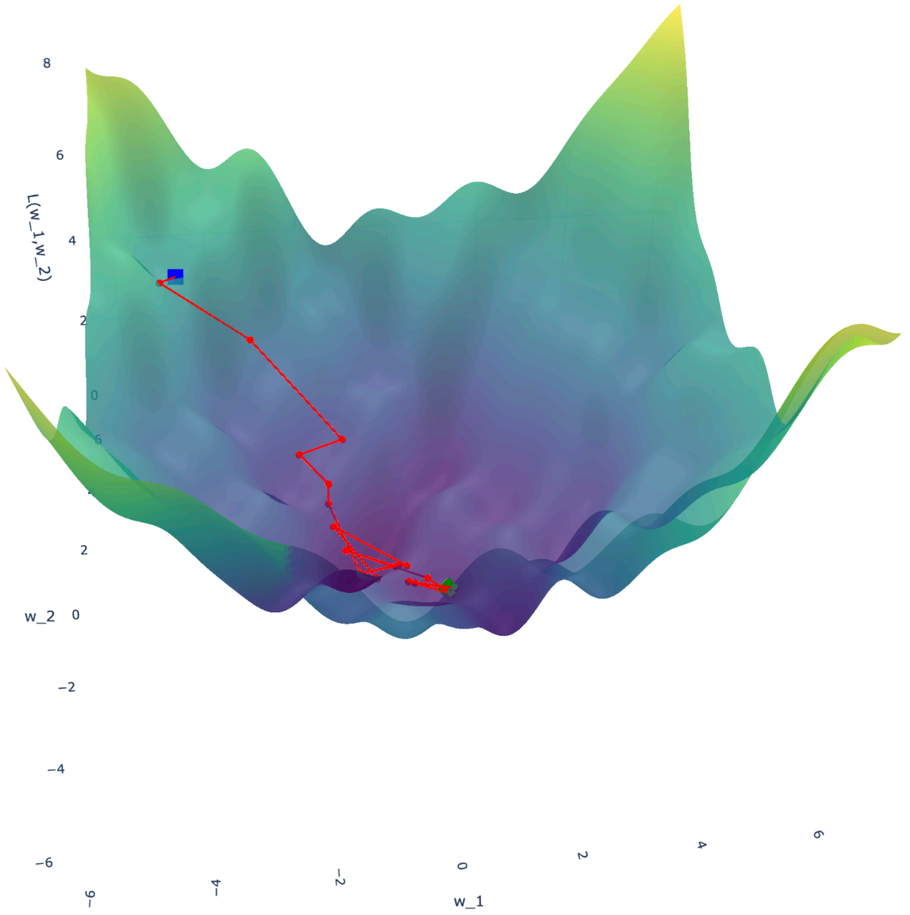
$$w^* = \arg\min_w \mathscr{L}(w)$$

$w = (w_1, \ldots, w_M) \in \mathbb{R}^M$ Parámetros del modelo

$\mathscr{L}(w) = \mathscr{L}(w \mid x, y)$ depende de los datos observados (x, y) y de los pesos.

$$\mathscr{L}(w) = \sum_{k=1}^{N} l(y_k, f_w(x_k))$$

$$w^{(i+1)} \leftarrow w^{(i)} - \eta_i \nabla_w \mathscr{L}(w^{(i)}) \rightarrow w_{t+1} \leftarrow w_t - \eta_i \nabla_w \mathscr{L}(w_t)$$



$$\frac{\partial \mathscr{L}(w_t)}{\partial w_{j,i}^{(l)}} = \sum_{k=1}^{N} \frac{\partial l(y_k, f_w(x_k))}{\partial f_w(x_k)} \left. \frac{\partial f_w(x_k)}{\partial w_{j,i}^{(l)}} \right|_{w=w_t}$$

$f_w(x_k)$ es una red neuronal de la forma

$$f_w(x_k) = (h_L \circ h_{L-1} \circ \cdots \circ h_1)(x_k)$$

con $h_l(x^{(l-1)}) = \sigma^{(l)}(W^{(l)} x^{(l-1)} + b^{(l)}) = \sigma^{(l)}(a^{(l)})$

Cálculo del gradiente: retropropagación

Por qué retropropagación?

Existen varias maneras de calcular el gradiente:

- Diferencias finitas: $\frac{\partial \mathscr{L}}{\partial w_i} \approx \frac{\mathscr{L}(w + h\mathbf{e}_i) - \mathscr{L}(w)}{h}$ (alto costo)

- A partir de la idea de grafo acíclico direccionado
  - hacia adelante
  - Hacia atrás

Cálculo del gradiente: retropropagación

$$\frac{\partial f_w(x_k)}{\partial w_{j,i}^{(l)}}\Big|_{w=w_t} \qquad f_w(x) = \sigma^{(L)}(a^{(L)})$$

La retropropagación lo que hace es descomponer eficientemente $\frac{\partial f_w(x_k)}{\partial w_{j,i}^{(l)}}$

usando la regla de la cadena por capa

$$a_j^{(l)} = \sum_{i=1}^{n_{l-1}} x_i^{(l-1)} w_{j,i}^{(l)} + b_j^{(l)}, \text{ (preactivación)} \qquad x_j^{(l)} = \sigma^{(l)}(a_j^{(l)}) \text{ (activación)}$$

$$\frac{\partial \mathscr{L}(w_t)}{\partial w_{j,i}^{(l)}} = \sum_{k=1}^{N} \frac{\partial l}{\partial a_j^{(l)}} \frac{\partial a_j^{(l)}}{\partial w_{j,i}^{(l)}}\Bigg|_{w=w_t} := \sum_{k=1}^{N} \delta_j^{(l)} \frac{\partial a_j^{(l)}}{\partial w_{j,i}^{(l)}}\Bigg|_{w=w_t}$$

$$\frac{\partial a_j^{(L)}}{\partial w_{j,i}^{(L)}} = x_i^{(L-1)}$$

$$\frac{\partial a_j^{(L)}}{\partial b_j^{(L)}} = 1$$

Cálculo del gradiente: retropropagación

Grafo computacional

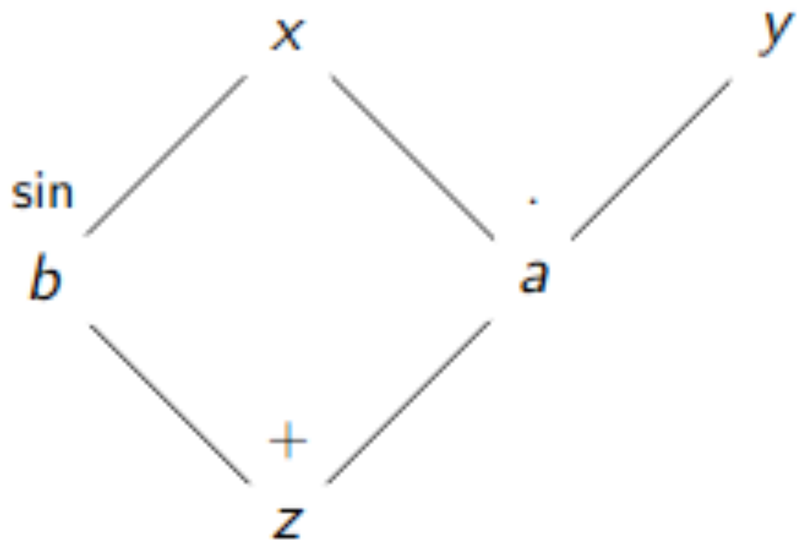
$$x \to a^{(1)} \to x^{(1)} \to a^{(2)} \to \cdots \to a^{(L)} \to x^{(L)} \to l$$

Luego loss.backward() retropropaga, recorriendo el grafo hacia atrás aplicando la regla de la cadena.

Backprop o retropropagación es el algoritmo matemático y **Autograd** o diferenciación automática es la implementación automática sobre un grafo computacional

Cálculo del gradiente: retropropagación

Grafo computacional **hacia delante:**



$$z(x, y) = xy + \sin(x)$$

Cómo afectan los cambios en x e y a la salida en z?

$$a = xy, \quad b = \sin(x), \quad z = a + b$$

$$da = y\,dx + x\,dy \qquad db = \cos(x)\,dx$$

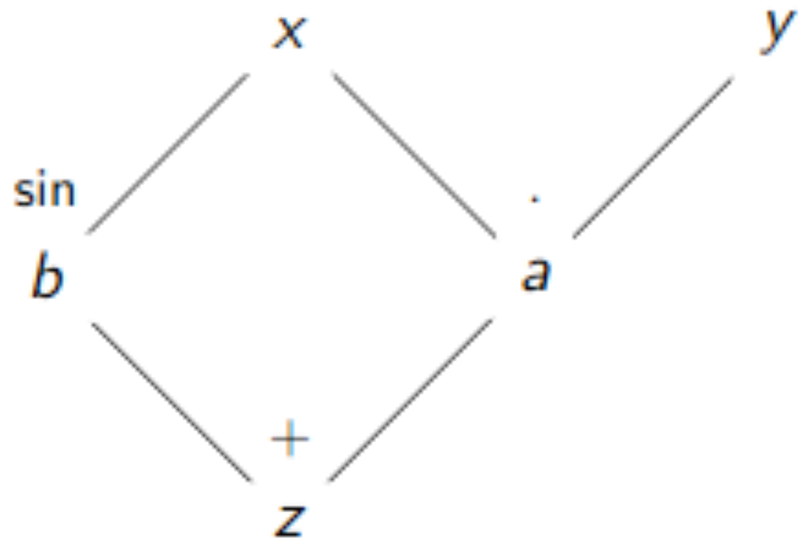
$$dz = da + db$$

```
# Modo directo: dx=1, dy=0 para derivada respecto a x
dx, dy = 1.0, 0.0
da = y * dx + x * dy
db = cos(x) * dx
dz = da + db
# dz contiene ahora la derivada parcial ∂z/∂x
```

Para calcular el gradiente completo con respecto a n parámetros, es necesario ejecutar n veces

Cálculo del gradiente: retropropagación

Grafo computacional **hacia atrás:**



$$z(x, y) = xy + \sin(x)$$

Cuál es el impacto de la salida z sobre cada elemento intermedio?

$$a = xy, \quad b = \sin(x), \quad z = a + b$$

$$\bar{v} := \frac{\partial z}{\partial v} \implies \bar{z} = 1$$

$$\bar{a} = \frac{\partial z}{\partial a} = \bar{z}\frac{\partial z}{\partial a} = \bar{z} \qquad \bar{b} = \frac{\partial z}{\partial b} = \bar{z}\frac{\partial z}{\partial b} = \bar{z}$$

```
# Modo inverso: retropropagación
bar_z = 1.0
bar_a = bar_z
bar_b = bar_z
bar_y = x * bar_a
bar_x = y * bar_a + cos(x) * bar_b
# bar_x y bar_y contienen ahora las derivadas parciales completas
```

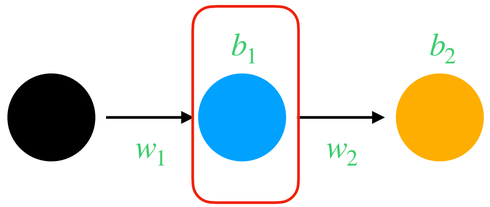
Una sola pasada hacia atrás produce todas las derivadas parciales

$$\bar{y} = \frac{\partial z}{\partial y} = \bar{a}\frac{\partial a}{\partial y} = \bar{a}\,x$$

$$\bar{x} = \frac{\partial z}{\partial x} = \bar{a}\frac{\partial a}{\partial x} + \bar{b}\frac{\partial b}{\partial x} = \bar{a}\,y + \bar{b}\,\cos(x)$$

Cálculo del gradiente: retropropagación

Ejemplo



$$a_1 = w_1 x + b_1$$

$$h = \tanh(a_1)$$

$$\hat{y} = w_2 h + b_2$$

$$L = \frac{1}{2}(\hat{y} - y)^2$$

Viene dato del batch: $(x_i, y_i) = (2,1)$

Incialmente (t=1): $w_1 = 0.5;\ b_1 = 0;\ w_2 = 0.4;\ b_2 = 0$

1. Hacemos forward pass: $a_1 = 1;\ h \approx 0.7616;\ \hat{y} \approx 0.3046;\ L \approx 0.2418$

2. Backprop:

$$\frac{\partial L}{\partial w_2} = (\hat{y} - y)h \approx -0.5296; \quad \frac{\partial L}{\partial b_2} = \hat{y} - y = -0.6954$$

$$\frac{\partial L}{\partial w_1} = (\hat{y} - y)w_2(1 - \tanh^2(a_1))x \approx -0.2336$$

$$\frac{\partial L}{\partial b_1} = (\hat{y} - y)w_2(1 - \tanh^2(a_1)) \approx -0.1168$$

$$w_{t+1} \leftarrow w_t - \eta_i \nabla_w \mathcal{L}(w_t)$$

Si $\eta = 0.1$, $w_1^{t=2} = 0.5 - 0.1(-0.2336) = 0.52336;\ b_1^{t=2} \approx 0.0117;\ w_2^{t=2} \approx 0.4530;\ b_2^{t=2} \approx 0.0695$

Cálculo del gradiente: Autograd

Autograd nos permite calcular mediante un grafo acíclico direccionado las derivadas parciales de una función parametrizada!!

Podríamos imponer condiciones sobre las derivadas de nuestra función en la función de costo… ?

Podríamos tal vez incluir ecuaciones diferenciales directamente en la función de costo…?

Laplace

$$\nabla^2 u = 0$$

Calor

$$\frac{\partial u}{\partial t} = \alpha \nabla^2 u$$

Schrödinger

$$i\hbar \frac{\partial \psi}{\partial t} = -\frac{\hbar^2}{2m} \nabla^2 \psi + V\psi$$

Poisson

$$\nabla^2 u = f(x)$$

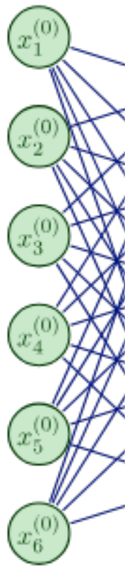
Onda

$$\frac{\partial^2 u}{\partial t^2} = c^2 \nabla^2 u$$

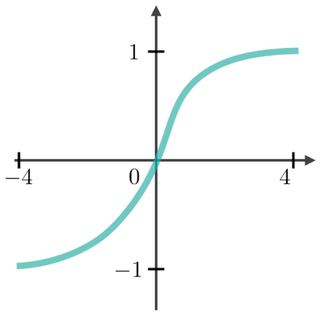
Problemas con el gradiente

Vanishing Gradient

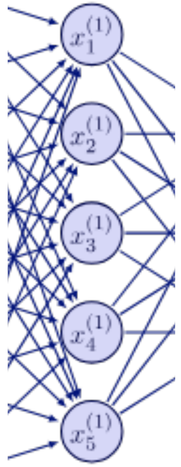
$$w_{1,1}$$



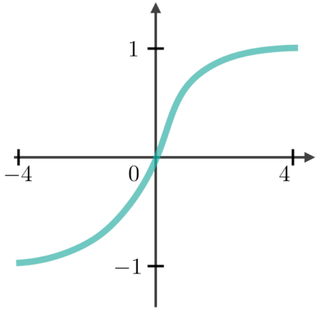
$$\frac{\partial a^{(1)}}{\partial w_{11}} \cdots$$



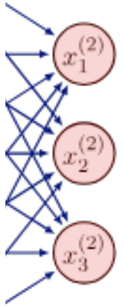
$$\frac{\partial h^{(L-2)}}{\partial a^{(L-3)}} \in [0,1]$$



$$\frac{\partial a^{(L-1)}}{\partial h^{(L-2)}}$$



$$\frac{\partial h^{(L-1)}}{\partial a^{(L-1)}} \in [0,1]$$



$$\frac{\partial a^{(L)}}{\partial h^{(L-1)}}$$

$$\frac{\partial l_w}{\partial a^{(L)}}$$

$$l_w(\hat{y}_i, y_i)$$

https://en.wikipedia.org/wiki/Vanishing_gradient_problem

Problemas con el gradiente: soluciones

Inicialización conveniente y BatchNorm

Elegir los pesos de forma que la varianza de activaciones y gradientes se mantenga estable.

$$\mathrm{Var}(h_l) = \mathrm{Var}(W_l x) \approx \mathrm{Var}(W_l)\mathrm{Var}(x)$$

Idea general

$$\mathrm{Var}(W_l) \approx \frac{1}{n}$$

$$W \sim \mathcal{U}\left(\pm\sqrt{\frac{6}{n_{in}+n_{out}}}\right)$$ Glorot o Xavier

$$W \sim \mathcal{N}\left(0, 2/n_{l-1}\right)$$ He o Kaiming

BatchNorm

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_B)^2$$

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

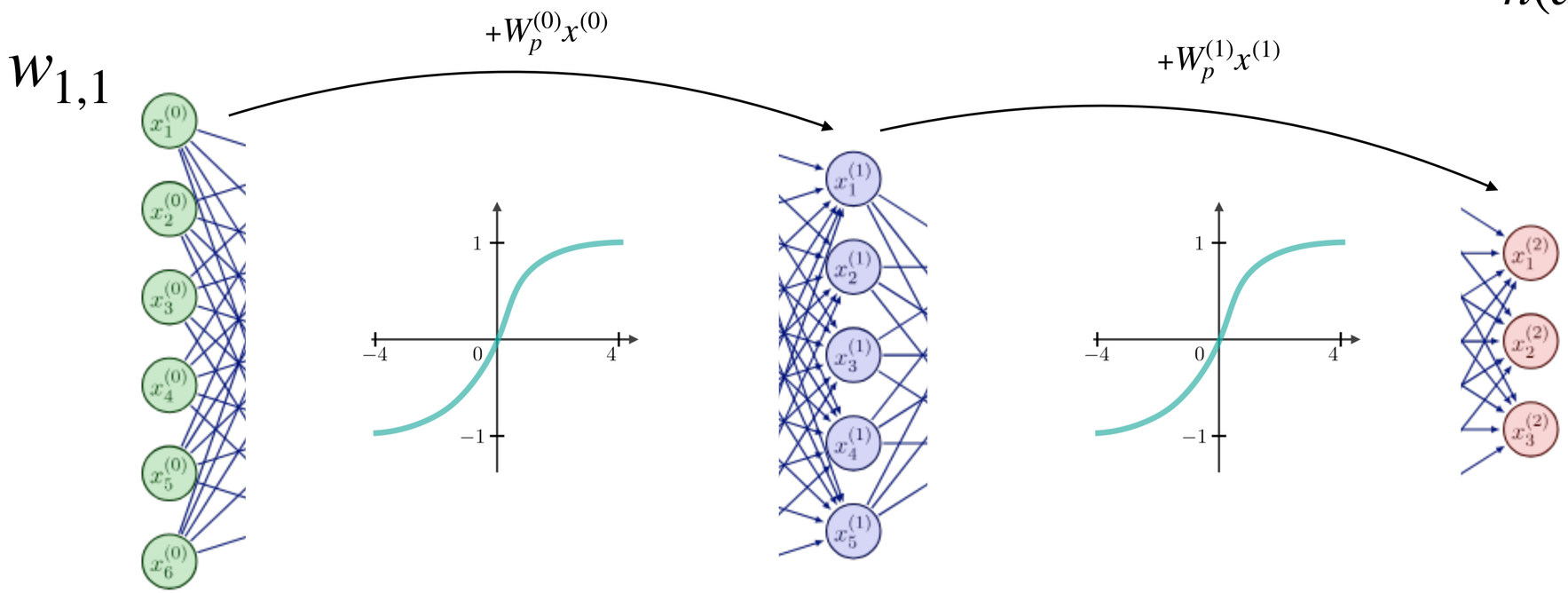
$$\tilde{x}_i = \gamma \hat{x}_i + \beta, \text{ con } \gamma, \beta$$

parámetros aprendibles por neurona

Problemas con el gradiente: soluciones

Conexiones residuales

$$\tilde{h}(a) = h(a) + \alpha a$$



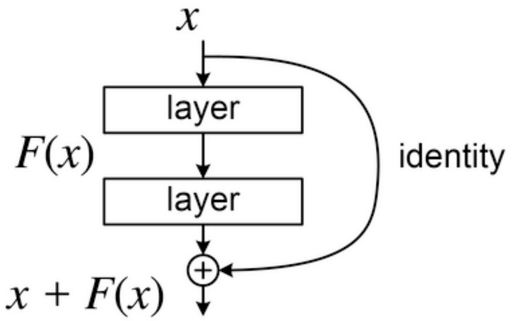
$$\frac{\partial a^{(1)}}{\partial w_{11}} \cdots \frac{\partial h^{(L-2)}}{\partial a^{(L-3)}} + \alpha \mathbf{1} \quad \frac{\partial a^{(L-1)}}{\partial \tilde{h}^{(L-2)}} \quad \frac{\partial h^{(L-1)}}{\partial a^{(L-1)}} + \alpha \mathbf{1} \quad \frac{\partial a^{(L)}}{\partial \tilde{h}^{(L-1)}} \quad \frac{\partial l_w}{\partial a^{(L)}}$$

$$l_w(\hat{y}_i, y_i)$$

https://en.wikipedia.org/wiki/Vanishing_gradient_problem

Conexiones residuales

$$f(x) = F(x) + x \rightarrow \nabla F + I$$

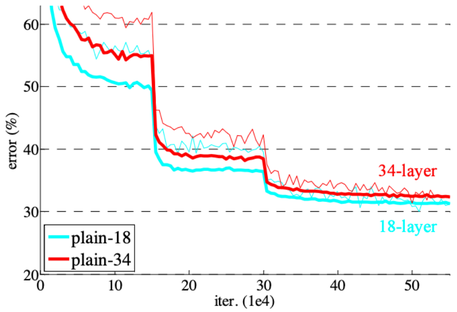


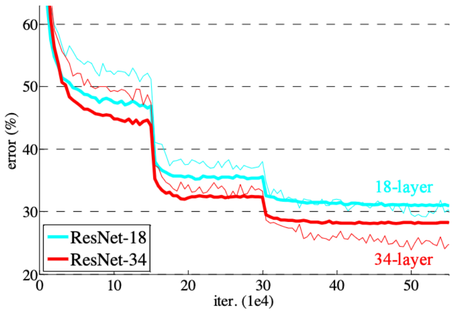
$$f_n \circ f_{n-1} \circ \cdots \circ f_1 \implies \nabla \text{ para capa } l \rightarrow I + \nabla F_{l+1} + \nabla F_{l+2} \nabla F_{l+1} + \cdots$$

Caso ejemplo: ResNet (2015)

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com





14 millones de imágenes

Etiquetadas a mano (2009)

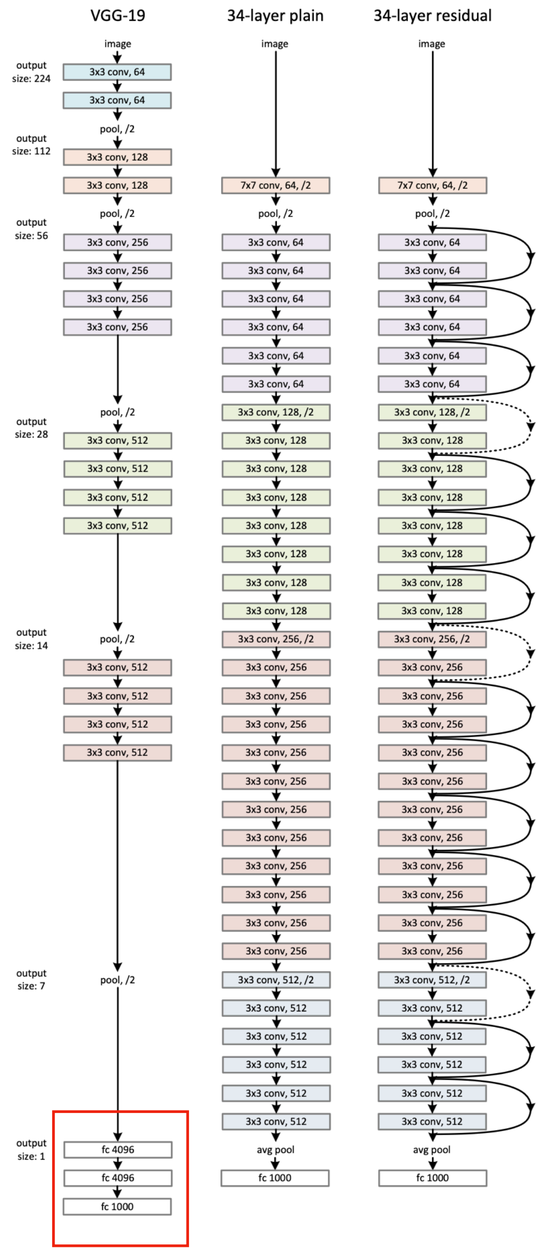
21841 etiquetas jerárquicas

AlexNet (2012)~15.3% (~60M)

model	top-1 err.	top-5 err.
VGG-16 [41]	28.07	9.33
GoogLeNet [44]	-	9.15
PReLU-net [13]	24.27	7.38
plain-34	28.54	10.02
ResNet-34 A	25.03	7.76
ResNet-34 B	24.52	7.46
ResNet-34 C	24.19	7.40
ResNet-50	22.85	6.71
ResNet-101	21.75	6.05
ResNet-152	**21.43**	**5.71**

Table 3. Error rates (%, **10-crop** testing) on ImageNet validation. VGG-16 is based on our test. ResNet-50/101/152 are of option B that only uses projections for increasing dimensions.

(2014)~7.3% (~144M) (2015)~5.7% (~21M)

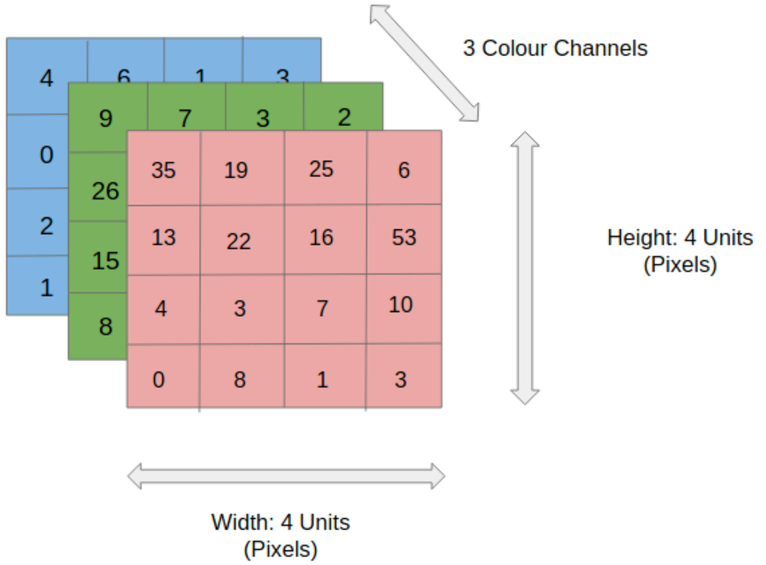


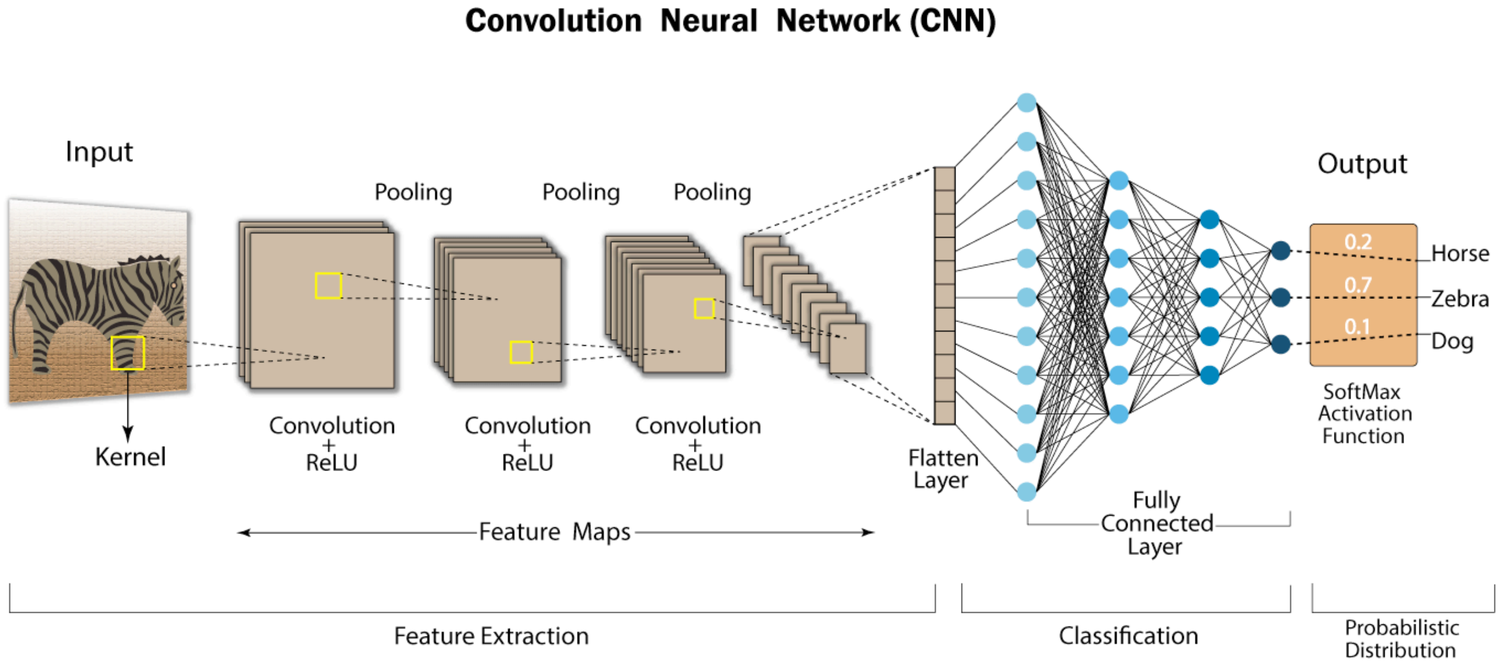
https://arxiv.org/abs/1512.03385

https://es.wikipedia.org/wiki/Red_neuronal_residual

Redes Neuronales Convolucionales

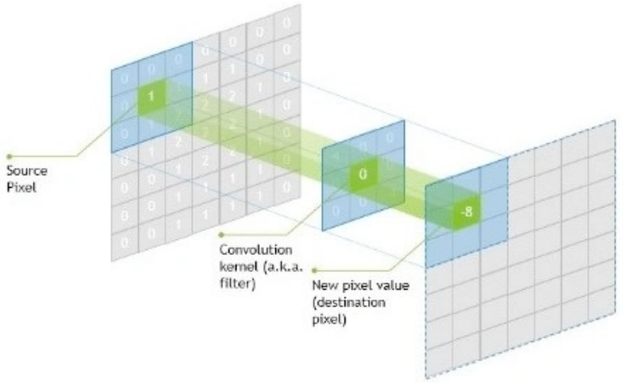
Introducción





Redes Neuronales Convolucionales

Convolución



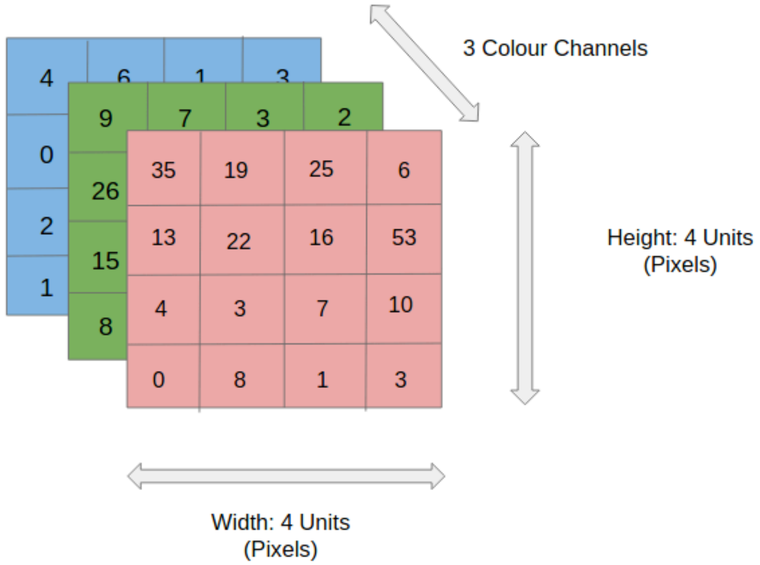
Imagen

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

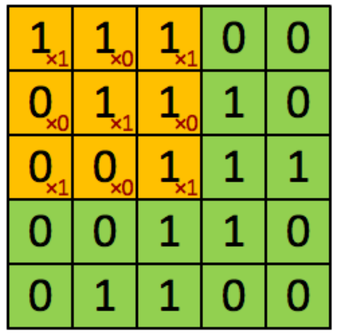
Kernel

1	0	1
0	1	0
1	0	1

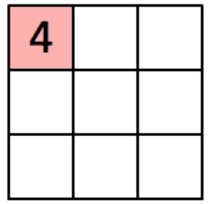
(N, C_{in}, H, W)



$$\text{out}(N_i, C_{out_j}) = \text{bias}(C_{out_j}) + \sum_{k=0}^{C_{in}-1} \text{weight}(C_{out_j}, k) \star \text{input}(N_i, k)$$



Image



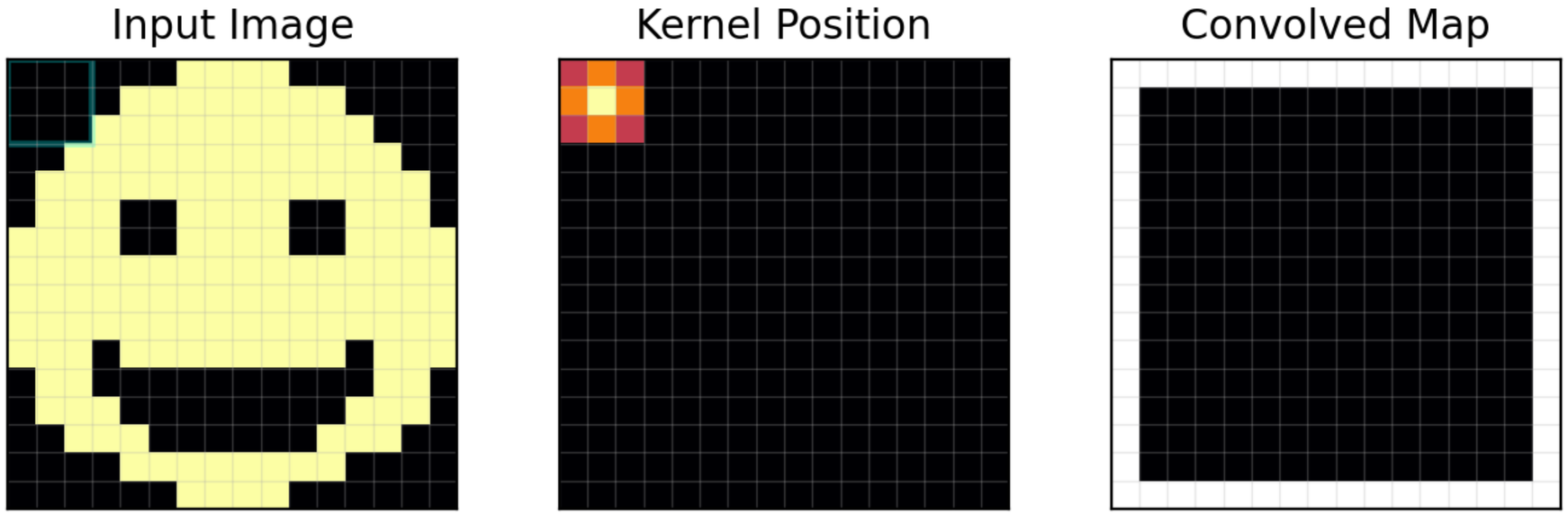
Convolved Feature

$(N, C_{\text{out}}, H_{\text{out}}, W_{\text{out}})$

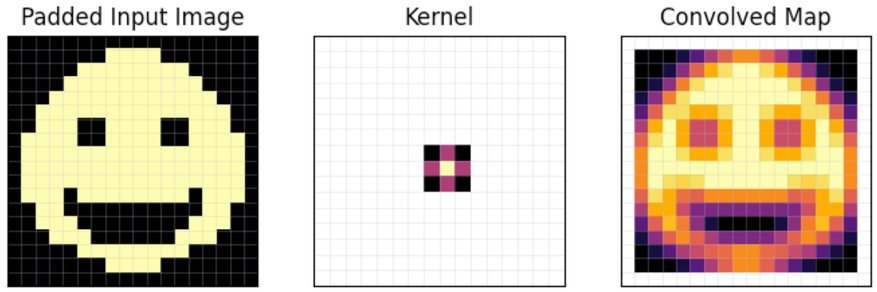
https://docs.pytorch.org/docs/stable/generated/torch.nn.Conv2d.html

Redes Neuronales Convolucionales

Convolución



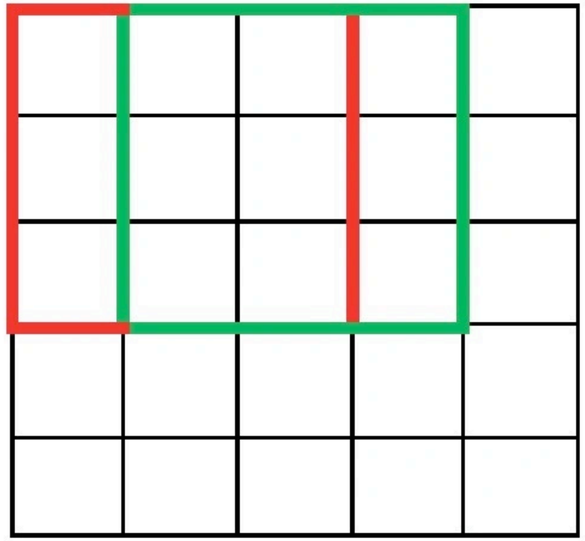
Padding



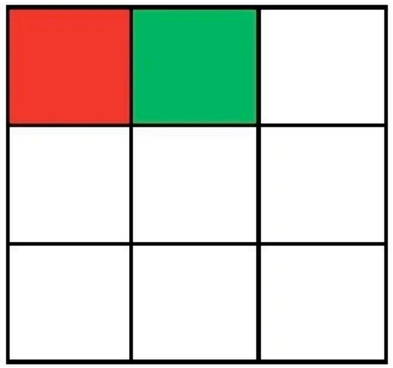
Redes Neuronales Convolucionales

Stride

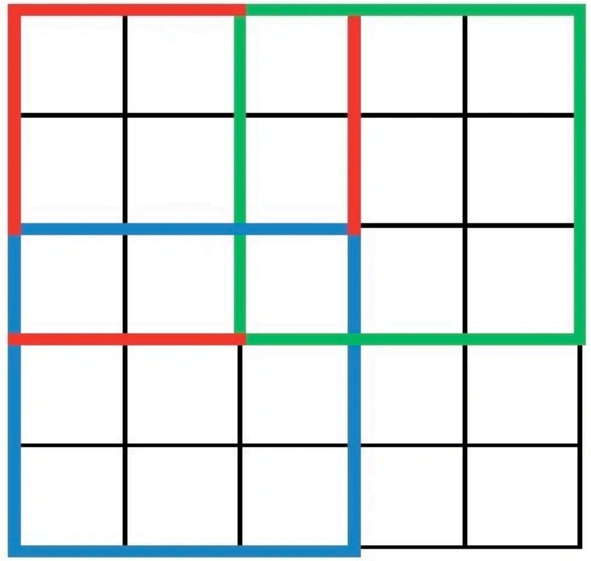
Convolution with Stride=1



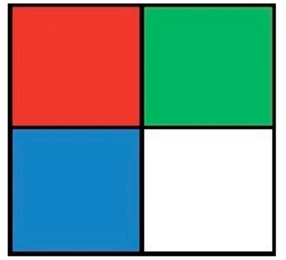
Output

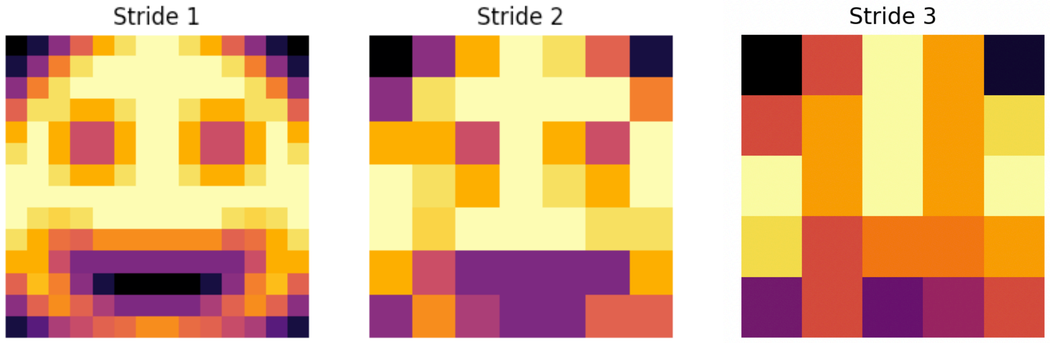


Convolution with Stride=2



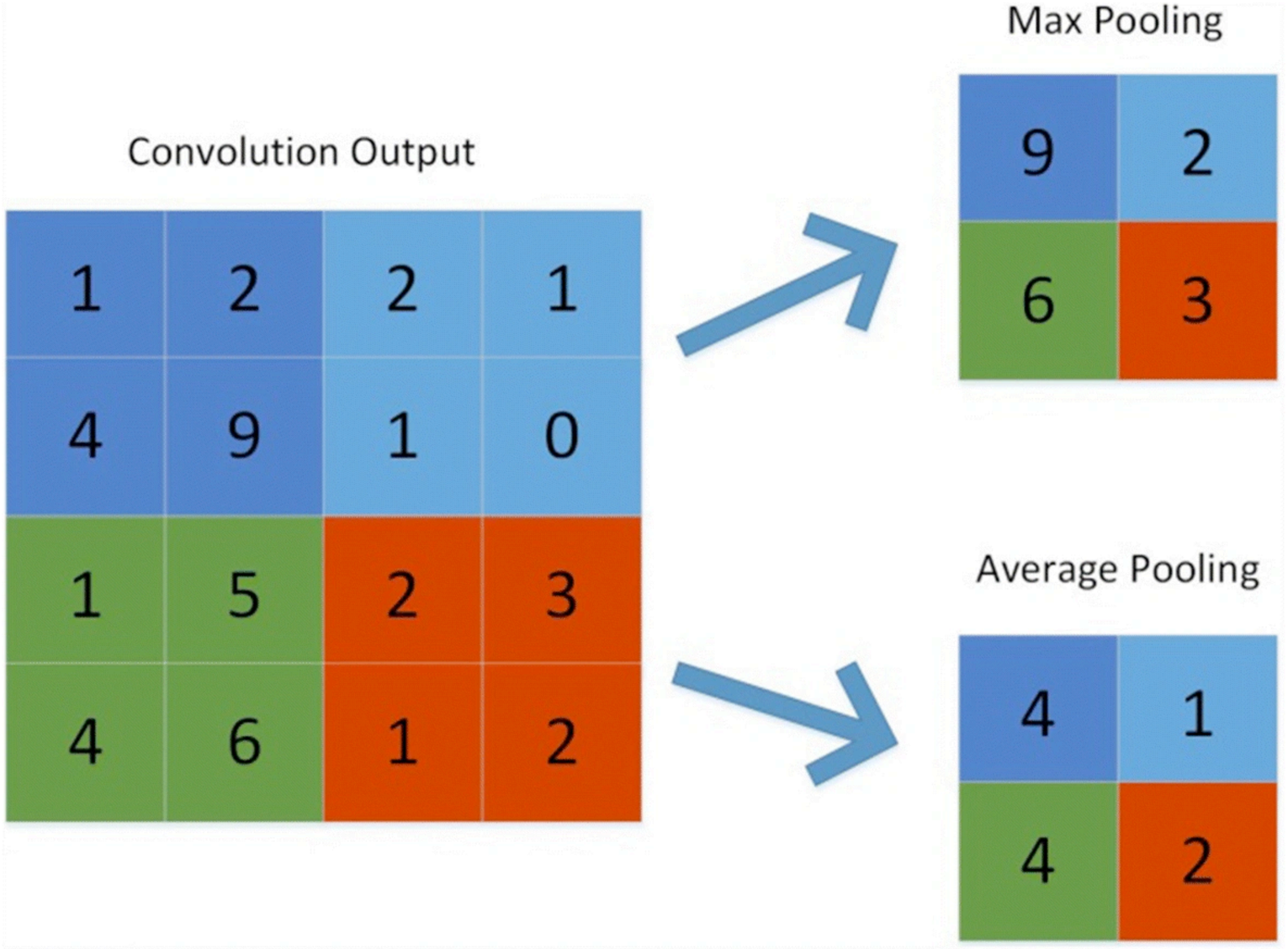
Output





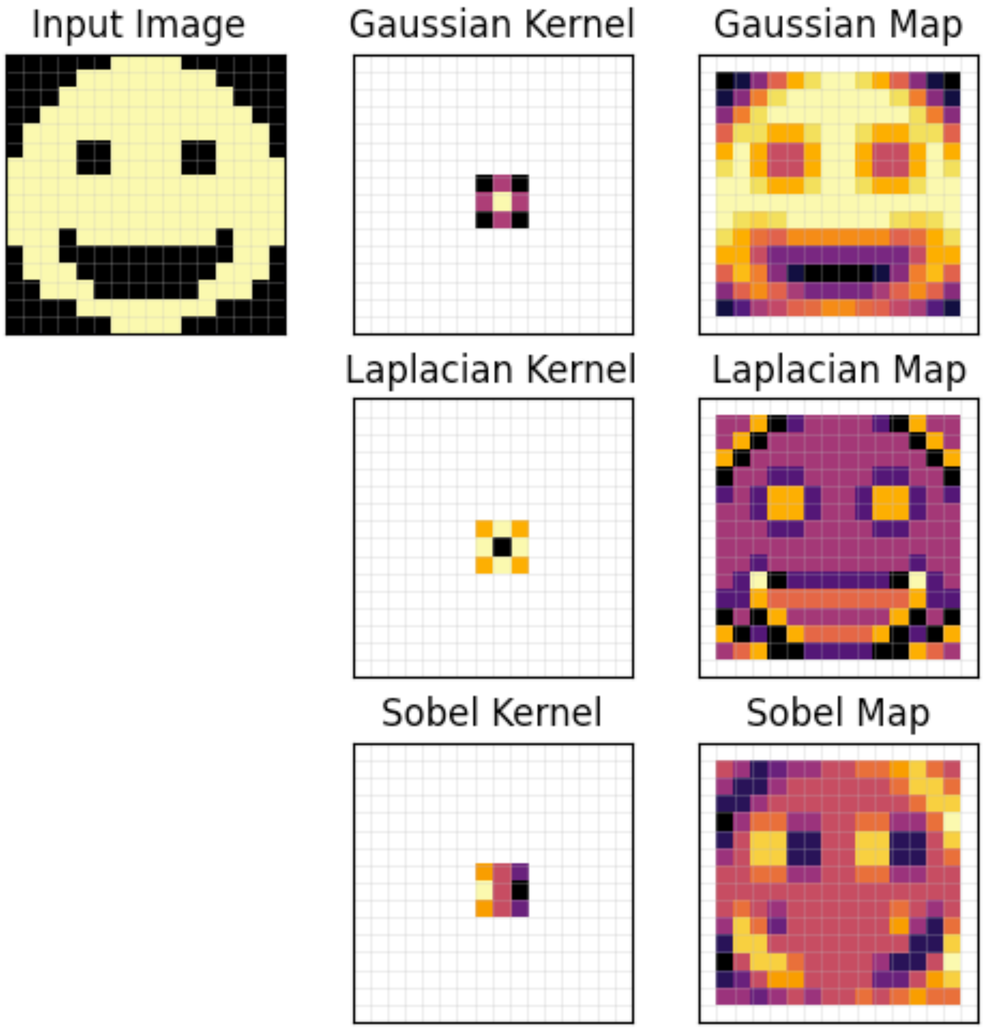
Redes Neuronales Convolucionales

Pooling



Redes Neuronales Convolucionales

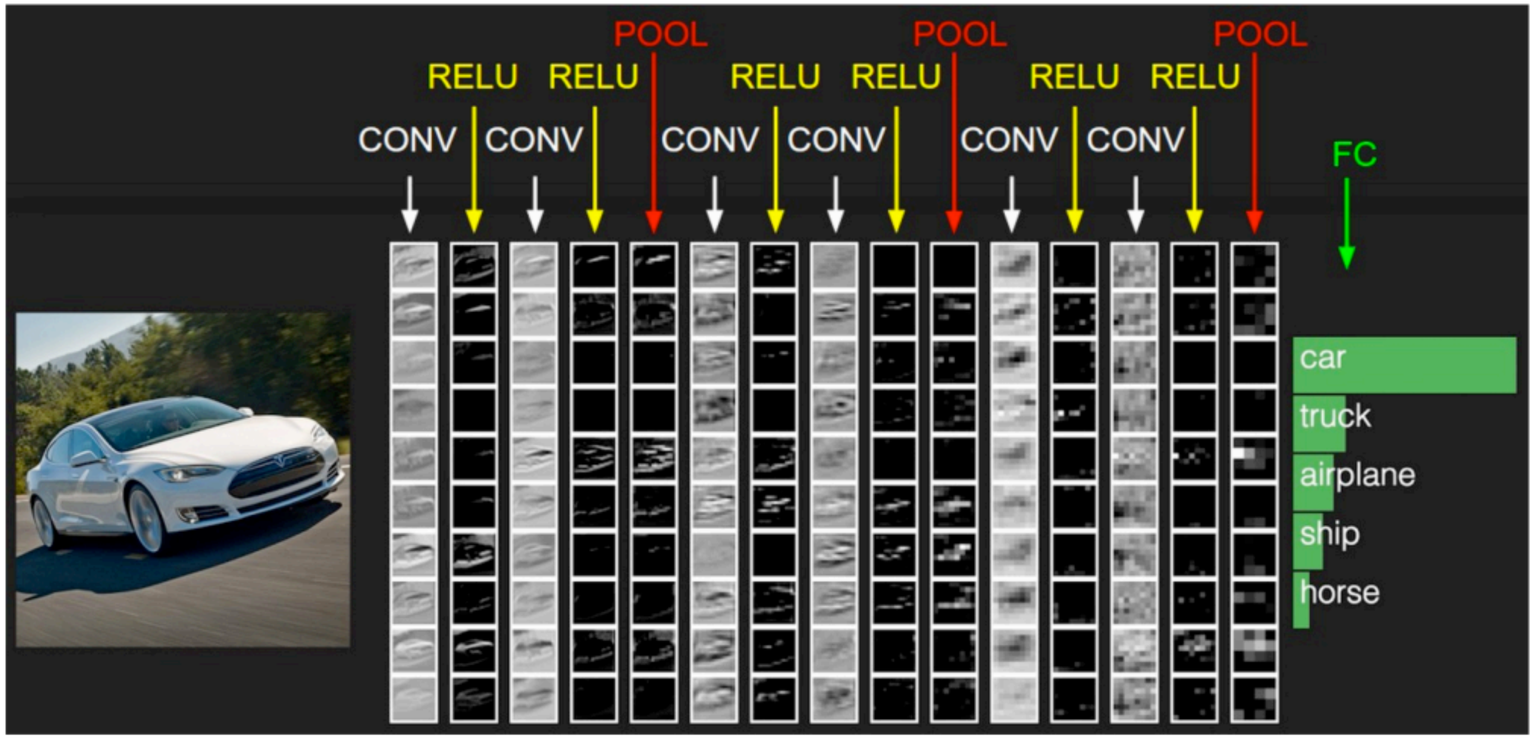
Sobre los kernels



https://poloclub.github.io/cnn-explainer/

Redes Neuronales Convolucionales

Clasificación



$$\mathscr{L}_{\text{CE}}(y, p) = -\sum_{k=1}^{K} y_k \log(p_k),$$

$$p_k = \frac{e^{z_k}}{\sum_{j=1}^{K} e^{z_j}}$$

https://poloclub.github.io/cnn-explainer/

Teórico práctico semana 2: CNN

Redes Neuronales Recurrentes

Utilidad: Datos secuenciales

- Generación de texto: “Qué buenas que están las RNN!”
- Análisis de sensaciones: “Me encantó esa película.” → positivo
- Traducción automática: “Me tomo el palo” → “I drink the stick”
- Secuencias: “ATGCGTACGTT” → clasificación genética
- Reconocimiento de voz: Audio.mp4 → “Soy cordobés, me gusta el vino y la joda”
- Predicción de series temporales: precios hasta tiempo t → precio a $t + 1$

Redes Neuronales Recurrentes

Idea general

En cada paso t temporal o de la secuencia:

- la red tiene en cuenta el valor de entrada actual $x^{<t>}$
- Tiene en cuenta el estado oculto anterior $a^{<t-1>}$
- Produce un nuevo estado oculto $a^{<t>}$

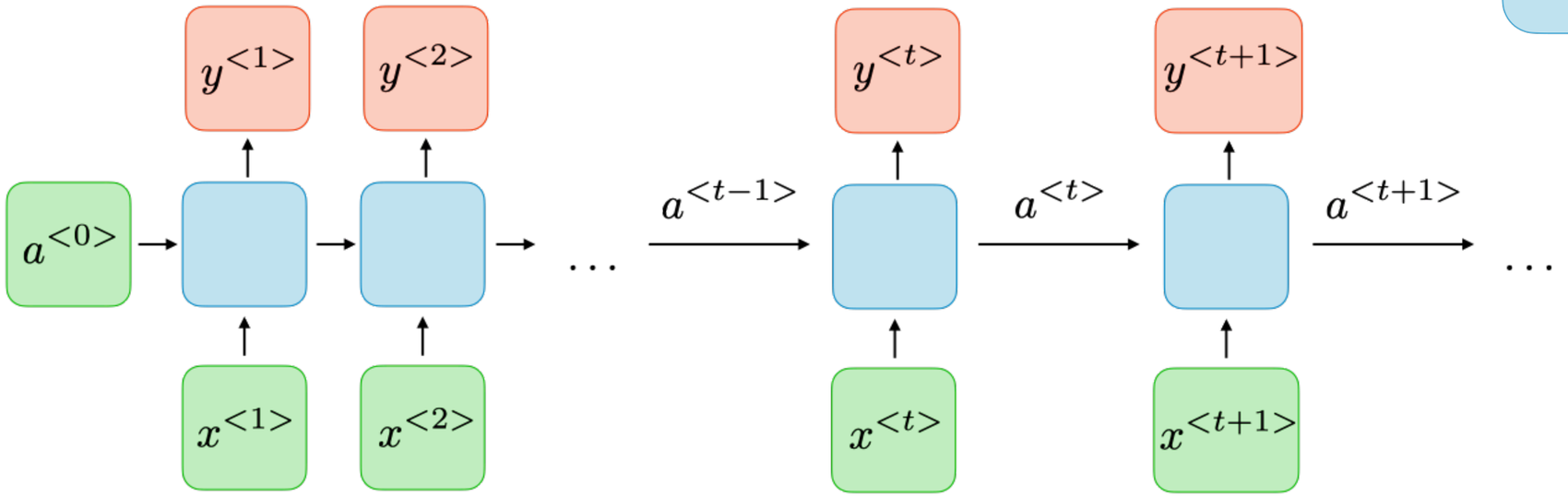
$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

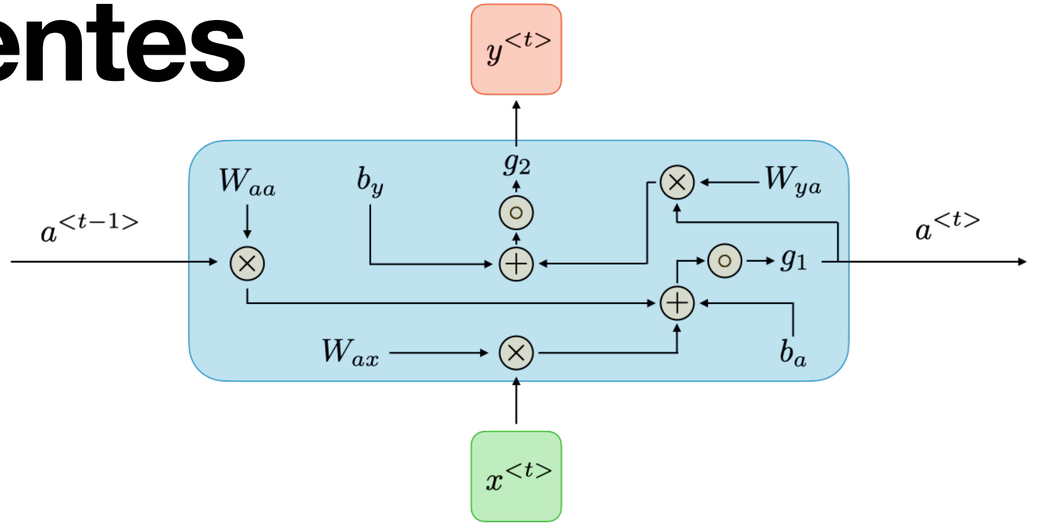
Representación de memoria

Redes Neuronales Recurrentes

Arquitectura

Many-to-many





$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

$$y^{<t>} = g_2(w_{ya}a^{<t>} + b_y)$$

Con $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ coeficientes compartidos temporalmente

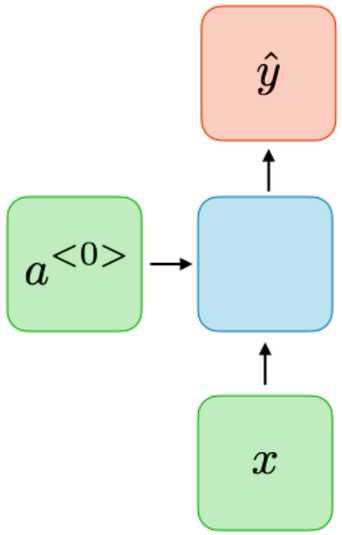
g_1, g_2: Funciones de activación

https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks/

Redes Neuronales Recurrentes

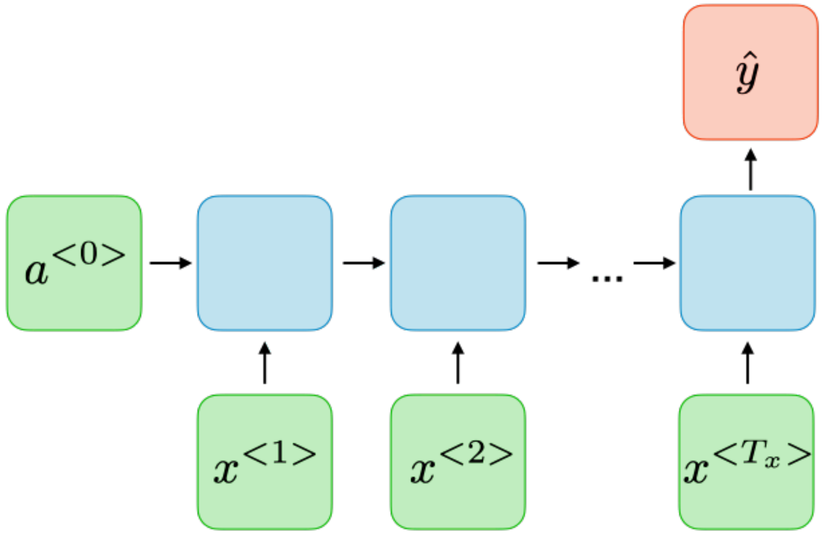
Tipos

One-to-one



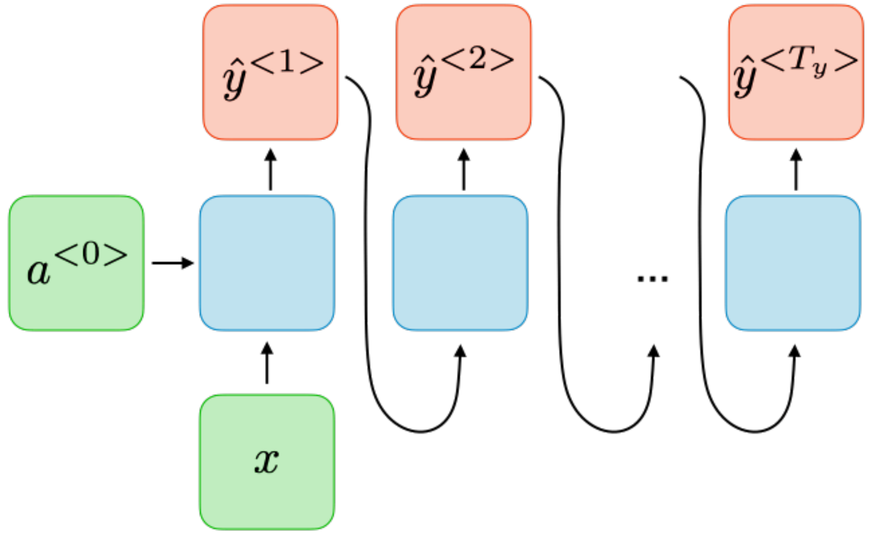
$T_x = T_y = 1$

Many-to-one



$T_x = 1;\ T_y > 1$

One-to-many



$T_x > 1;\ T_y = 1$

Redes Neuronales Recurrentes

Función de costo

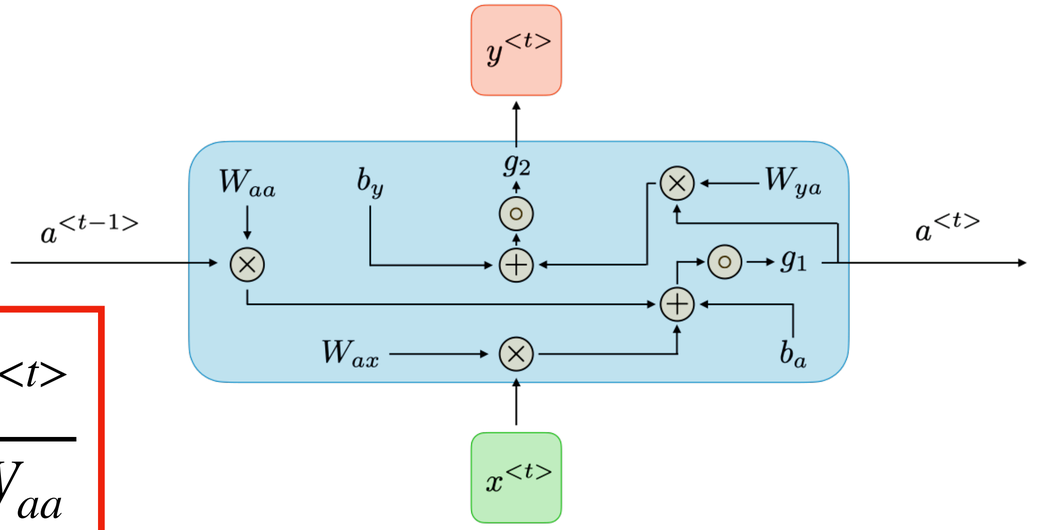
$$\mathscr{L}^{(T)}(y,\hat{y}) = \frac{1}{T}\sum_{t=1}^{T}\mathscr{L}^{<t>}(y^{<t>},\hat{y}^{<t>})$$

$$y^{<t>} = g_2(w_{ya}a^{<t>} + b_y)$$

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a)$$

El estado oculto $a^{<t>}$ afecta a $\mathscr{L}^{<t>}$ y a todas las funciones de costo futuras a tiempos $t+1, t+2, \ldots, T$

$$\frac{\partial \mathscr{L}^{(T)}}{\partial W_{ya}} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial \mathscr{L}^{<t>}}{\partial W_{ya}}$$



$$\frac{\partial \mathscr{L}^{(T)}}{\partial W_{ax}} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial \mathscr{L}^{(T)}}{\partial a^{<t>}}\frac{\partial a^{<t>}}{\partial W_{ax}} \qquad \frac{\partial \mathscr{L}^{(T)}}{\partial W_{aa}} = \frac{1}{T}\sum_{t=1}^{T}\frac{\partial \mathscr{L}^{(T)}}{\partial a^{<t>}}\frac{\partial a^{<t>}}{\partial W_{aa}}$$

Redes Neuronales Recurrentes

Problemas en RNNs

- Todo paso temporal sobreescribe el paso previo.
- No hay mecanismo explícito para recordar u olvidar información
- El gradiente explota o desaparece…
- Dependencias a largo T son difíciles de aprender.

Redes Neuronales Recurrentes

GRU vs. LSTM

$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b)$$

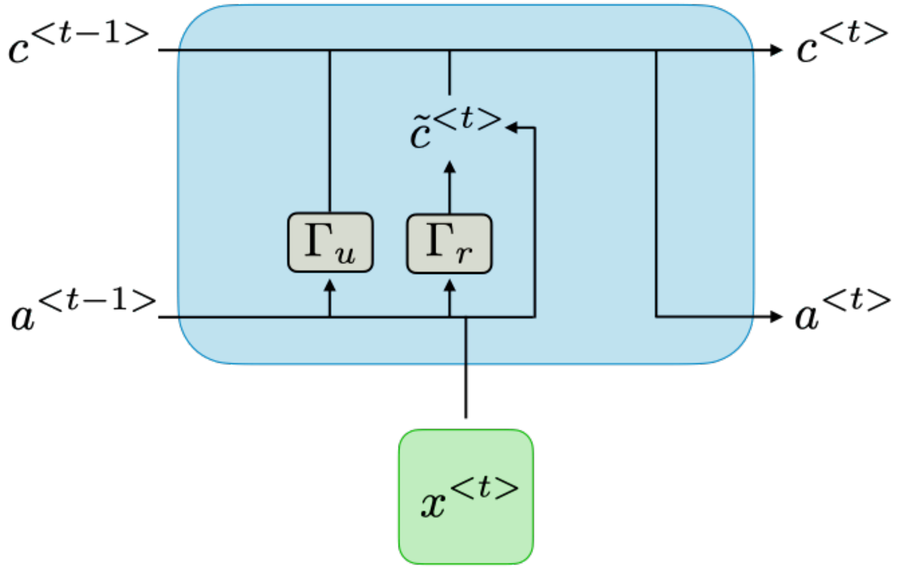
Γ_f = forget gate

Γ_u = update gate

Γ_r = reset gate

Γ_o = output gate

Gated Recurrent Unit

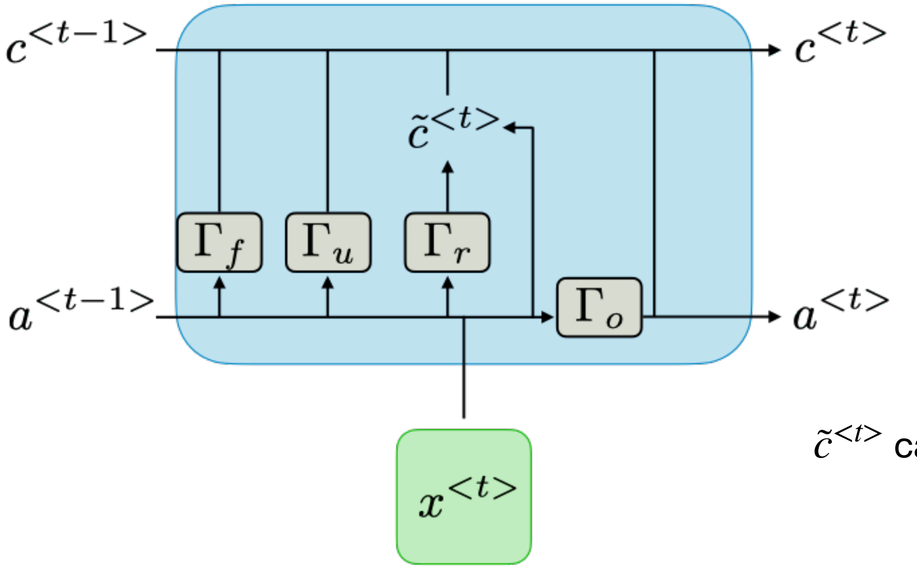


$$a^{<t>} = c^{<t>}$$

$$c^{<t>} = \Gamma_u \star \tilde{c}^{<t>} + (1 - \Gamma_u) \star c^{<t-1>}$$

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$$

Long Short Term Memory



Cell state

Hidden state

$\tilde{c}^{<t>}$ candidate state

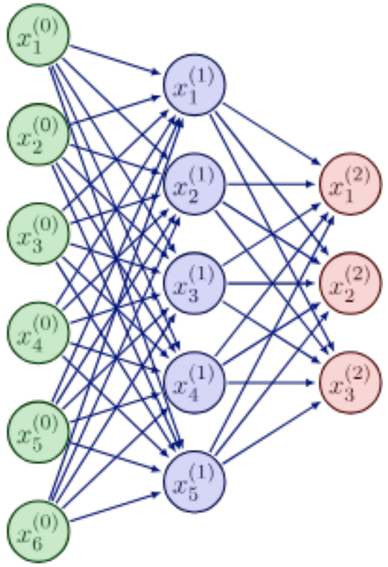
$$a^{<t>} = \Gamma_o \star c^{<t>}$$

$$c^{<t>} = \Gamma_u \star \tilde{c}^{<t>} + \Gamma_f \star c^{<t-1>}$$

$$\tilde{c}^{<t>} = \tanh(W_c[\Gamma_r \star a^{<t-1>}, x^{<t>}] + b_c)$$

Teórico práctico semana 2: Regresión y GRU

Sesgos inductivos y constraints físicos



→ Buenos aproximados universales → **Sólo aprenden de los datos**

Cómo incluímos conocimiento previo del problema en el modelo?

Un **sesgo inductivo** es cualquier restricción o estructura que guía el aprendizaje hacia soluciones plausibles.

Ya vimos algunos… se les ocurren?

Sesgos inductivos y constraints físicos

Formas de incorporar sesgos inductivos:

- Generar datos que respeten la física
- Aumentar datos usando simetrías (rotaciones, traslaciones, etc.)
- Diseñar arquitectura para que respete ciertas propiedades por construcción
- Agregar términos que penalicen violaciones físicas

Qué ganamos?

- Reducimos el espacio de soluciones
- Mejora la capacidad de generalización de nuestro modelo
- Ta vez necesitemos menos datos!

Sesgos inductivos y constraints físicos

Algunos ejemplos

Positividad de solución $\rightarrow y = \text{softplus}(z) = \frac{1}{\beta}\log(1+\exp(\beta z))$

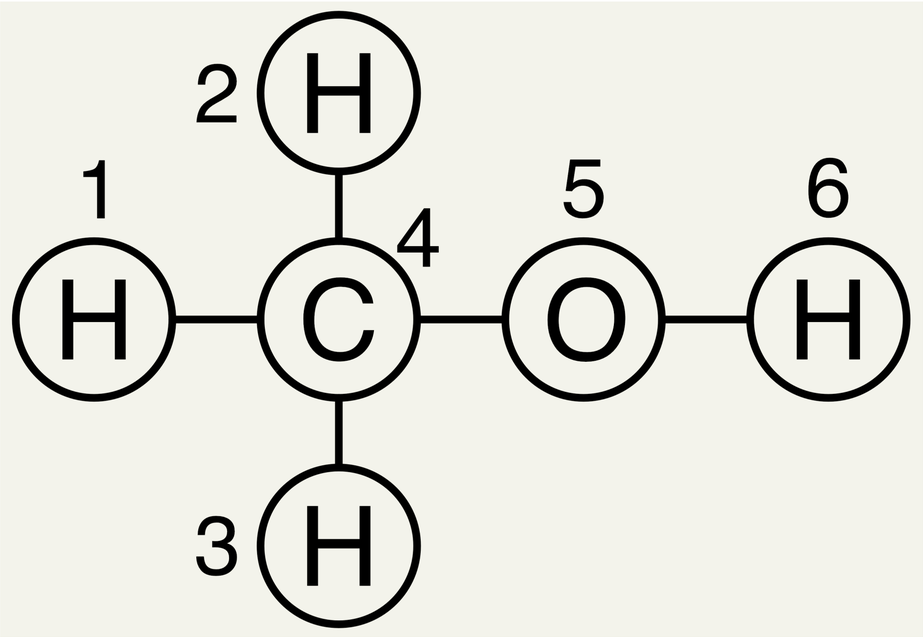
Conservaciones simples $\rightarrow \sum_{i=1}^{N} m_i = M \implies m_i = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} M$

Invariancias $\rightarrow$ inputs relativos, uso de redes equivariantes donde $f(g(x)) = g(f(x))$

Monotonicidad $\rightarrow \frac{dy}{dx} \geq 0$

Graph Neural Networks

Equivarianzas



Matriz de adyacencias con C=1

	1	2	3	4	5	6
1	0	1	1	1	1	0
2	1	0	0	0	0	0
3	1	0	0	0	0	0
4	1	0	0	0	1	0
5	1	0	0	0	0	1
6	0	0	0	0	1	0

Matriz de nodos **V**

Node	C	H	O
1	0	1	0
2	0	1	0
3	0	1	0
4	1	0	0
5	0	0	1
6	0	1	0

Matriz de adyacencias **E** con C=4

	1	2	3	4	5	6
1	0	0	0	1	0	0
2	0	0	0	1	0	0
3	0	0	0	1	0	0
4	1	1	1	0	1	0
5	0	0	0	1	0	1
6	0	0	0	0	1	0

$$f_k = \sigma\left(\sum_i \sum_j v_{ij} w_{jk}\right) \qquad v_{il} = \sigma\left(\frac{1}{d_i} e_{ij} v_{jk} w_{kl}\right)$$

https://dmol.pub/dl/gnn.html#representing-a-graph

Sesgos inductivos y constraints físicos

Conclusión

Al menos tres formas directas de incluir física:

- **datos**: qué observa el modelos y qué puede aprender de ello?
- **Arquitectura**: soft-constraints y hard-constraints que imponemos
- **Función de costo**: Qué pedimos que se debe cumplir?

Práctica 2