

# Detalles sobre Deep Learning

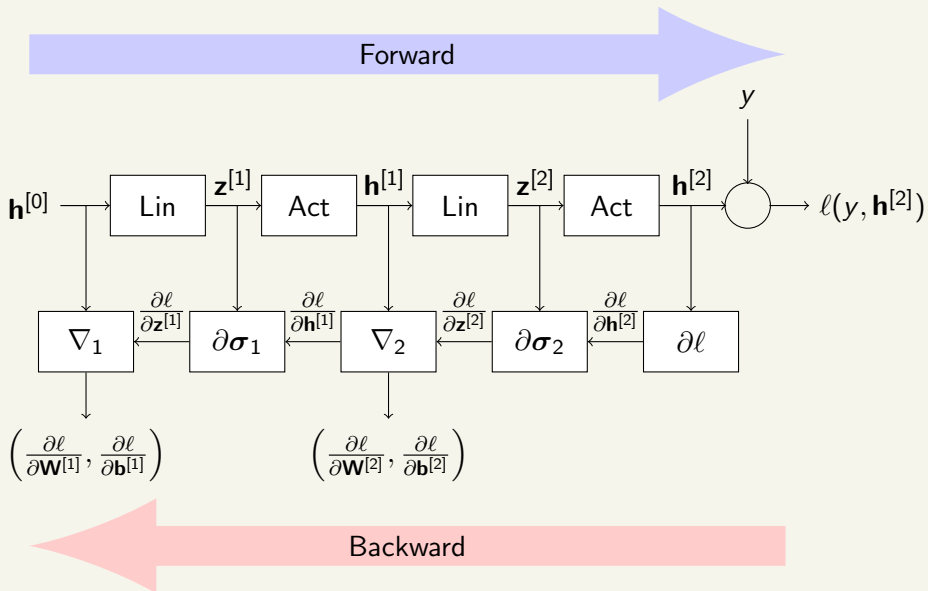
**Matias Vera - Juan Zuloaga - Lautaro Estienne**

Centro de Simulación Computacional para Aplicaciones Tecnológicas

# Agenda

- 1 Vanishing/Exploting
- 2 Tuneo de hiperparámetros
- 3 Underfitting y Overfitting
- 4 Regularización
  - Weight-decay
  - Dropout
  - Data augmentation
  - Early stopping
  - Autoencoders como regularización

# Forward-Backward Propagation



# Vanishing/Exploting

## Idea

Para redes muy profundas, la composición de layers puede generar magnitudes muy pequeñas o muy grandes que estropee la precisión numérica. Este efecto puede ocurrir tanto en las magnitudes (forward) o en sus gradientes (backward).

# Comparación sigmoide vs ReLU

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

- No explota  $\sigma(z) < 1$
- No explota su gradiente  
 $\sigma'(z) = \sigma(z) (1 - \sigma(z)) \leq \frac{1}{4}$
- Su gradiente se desvanece con mucha facilidad

$$\sigma(z) = \max\{0, z\}$$

- Su gradiente no desvanece ni explota  
 $\sigma'(z) = \mathbb{1}\{z > 0\}$ .
- Tiene tendencia a explotar  
 $\sigma(z) > z$
- Es computacionalmente más simple de computar
- Puede matar neuronas:  
 $(\mathbf{h}^{[l]})_k = 0$  para todo  $\mathbf{h}^{[l-1]}$   
(cambiarla por Leaky-Relu)

## ¿Como evitar la explosión de la ReLU?

Una manera de atenuar una posible explosión es con una inicialización adecuada. Sea  $\mathbf{b}^{[l]} = \mathbf{0}$  y que  $(\mathbf{W}^{[l]})_{ij} \sim \mathcal{N}(0, v)$  i.i.d. Tomemos una neurona:  $z^{[l]} = \sum_{k=1}^{n^{[l-1]}} w_k \sigma(z_k^{[l-1]})$  y supongamos que todos los  $z_k^{[l-1]}$  son idénticamente distribuidos con una densidad simétrica en el origen.

$$\begin{aligned}\text{var}(z^{[l]}) &= \mathbb{E} \left[ (z^{[l]})^2 \right] \\&= \sum_{i=1}^{n^{[l-1]}} \sum_{j=1}^{n^{[l-1]}} \mathbb{E}[w_i w_j] \mathbb{E} \left[ \sigma(z_i^{[l-1]}) \sigma(z_j^{[l-1]}) \right] \\&= \sum_{k=1}^{n^{[l-1]}} v \mathbb{E} \left[ \sigma^2(z_k^{[l-1]}) \right] \\&= n^{[l-1]} v \mathbb{E} \left[ (z^{[l-1]})^2 \mathbb{1} \{z^{[l-1]} > 0\} \right] \\&= \frac{n^{[l-1]} v}{2} \mathbb{E} \left[ (z^{[l-1]})^2 \right]\end{aligned}$$

# Inicialización de parámetros

$$\mathbb{E} \left[ \left( z^{[l]} \right)^2 \right] = \frac{n^{[l-1]} v}{2} \mathbb{E} \left[ \left( z^{[l-1]} \right)^2 \right]$$

Para mantener la variabilidad se propone  $(\mathbf{W}^{[l]})_{i,j} \sim \mathcal{N} \left( 0, \frac{2}{n^{[l-1]}} \right)$  para matrices que multiplican salidas de ReLU. (He Normal Initialization)

Otras inicializaciones:

- $(\mathbf{W}^{[l]})_{i,j} \sim \mathcal{N} \left( 0, \frac{1}{n^{[l-1]}} \right)$  (Lecun Normal Initialization)
- $(\mathbf{W}^{[l]})_{i,j} \sim \mathcal{N} \left( 0, \frac{2}{n^{[l-1]} + n^{[l]}} \right)$  (Glorot Normal Initialization)
- $(\mathbf{W}^{[l]})_{i,j} \sim \mathcal{U} \left( -\sqrt{\frac{6}{n^{[l-1]} + n^{[l]}}}, \sqrt{\frac{6}{n^{[l-1]} + n^{[l]}}} \right)$  (Glorot Uniform Initialization)
- Normales truncadas (si excede 2 desvíos va de nuevo)

---

*He et al. 2015:* "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification".

*Glorot-Bengio 2010:* "Understanding the difficulty of training deep feedforward neural networks"

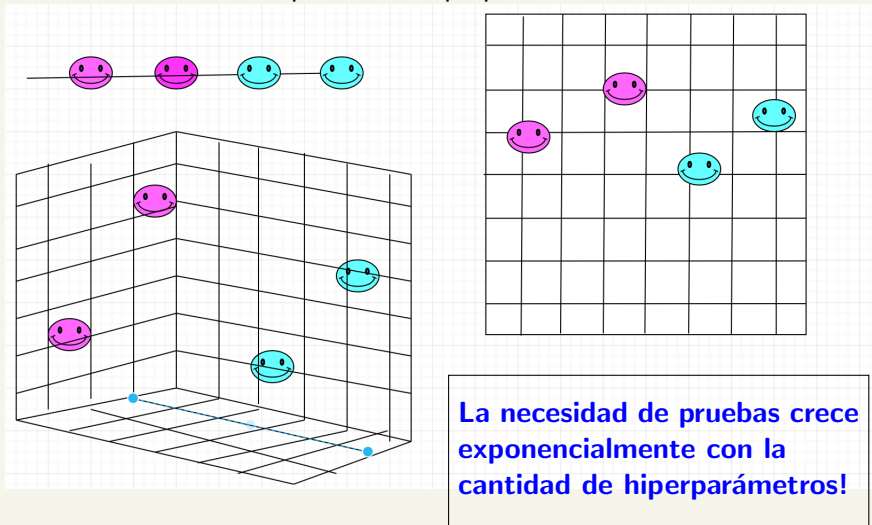
# Agenda

- 1 Vanishing/Exploting
- 2 Tuneo de hiperparámetros
- 3 Underfitting y Overfitting
- 4 Regularización
  - Weight-decay
  - Dropout
  - Data augmentation
  - Early stopping
  - Autoencoders como regularización



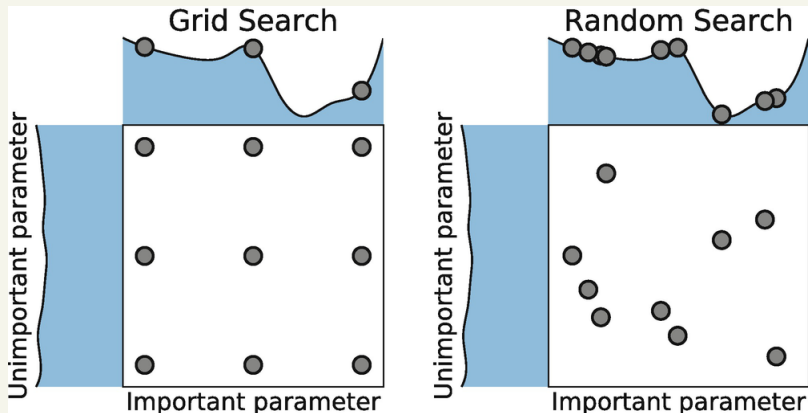
# La maldición de la dimensionalidad

La maldición también aplica a los hiperparámetros



# Búsqueda aleatoria

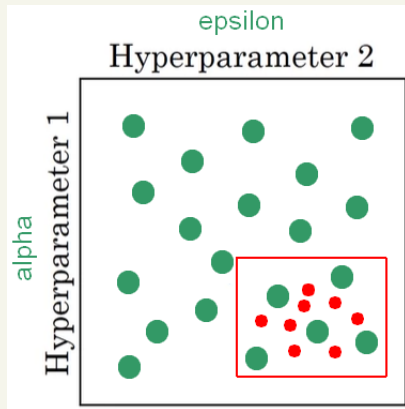
No todos los hiperparámetros son igual de importantes (ej.  $\alpha$  y  $\epsilon$ )



Random search nos permite variar muchas veces todos los parámetros.

# Búsqueda aleatoria

Hacerlo por etapas permite aprovechar más las simulaciones.



Simulo unos pocos puntos, veo donde está andando mejor y vuelvo a simular dentro de ese entorno.

# Sensibilidad

La sensibilidad de los parámetros no necesariamente es uniforme. Para la cantidad de layers o la cantidad de unidades ocultas parece razonable elegir al azar (equiprobable), pero para otros parámetros no es así.

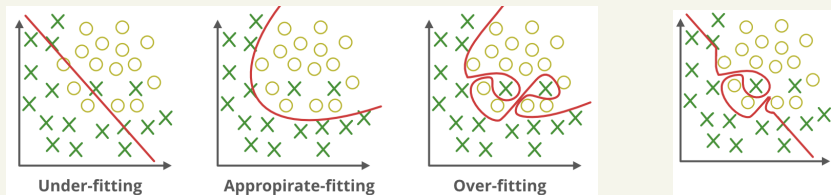
Algunos ejemplos:

- Learning Rate  $\alpha$ : Supongamos que decido que  $\alpha \in [0.0001, 1]$ . En este caso se suele simular  $\alpha = 10^{-4U}$  con  $U \sim \mathcal{U}(0, 1)$ .
- Weighted average rate  $\beta$ : Supongamos que decido que  $\beta \in [0.9, 0.999]$ . En este caso se suele simular  $\beta = 1 - 10^{-4U+1}$  con  $U \sim \mathcal{U}(0, 1)$ .

# Agenda

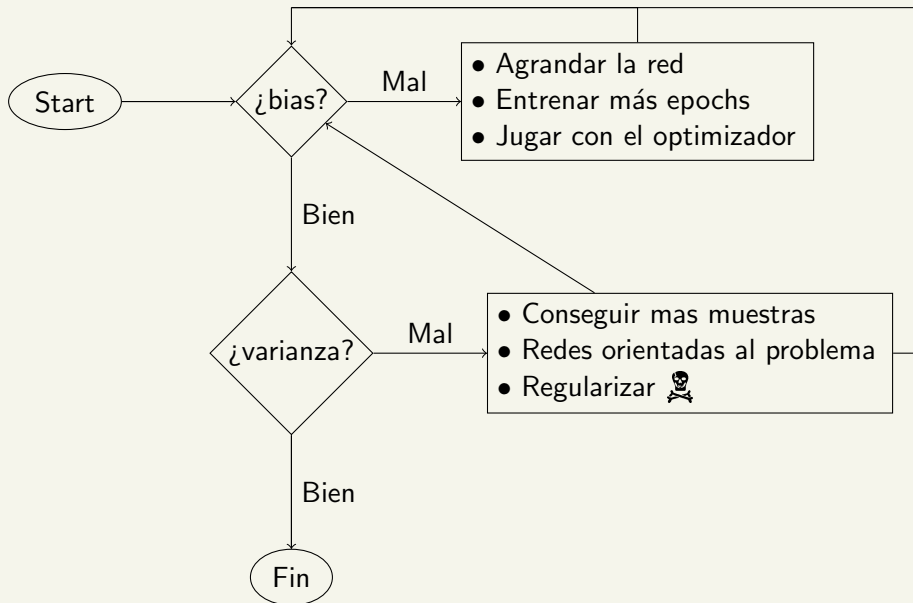
- 1 Vanishing/Exploting
- 2 Tuneo de hiperparámetros
- 3 Underfitting y Overfitting
- 4 Regularización
  - Weight-decay
  - Dropout
  - Data augmentation
  - Early stopping
  - Autoencoders como regularización

# Underfitting y Overfitting



|                  |          |                            |                            |      |
|------------------|----------|----------------------------|----------------------------|------|
| Train Error      | 1%       | 15%                        | 15%                        | 0.5% |
| Validation Error | 11%      | 16%                        | 30%                        | 1%   |
| Problema         | Varianza | Sesgo<br>(opt $\approx$ 0) | Ambos<br>(opt $\approx$ 0) | ✓    |

# Propuesta de algoritmo (primeras iteraciones)



# Atacar el punto débil (refinamiento)

¿Que conviene corregir? ¿Sesgo o varianza?

- **Avoidable bias:** Error de train - Error bayesiano
- **Generalization Gap:** Error de validación - Error de train

Ambos problemas se atacan por separado!



# Tipos de regularizaciones (algunos)

## ¿Cómo mejorar la capacidad de generalización del algoritmo?

- **Término de penalización (o regularizador):** Incluir un término adicional en la función costo.
- **Inyección de ruido:** Incluir algún tipo de ruido en el proceso de aprendizaje.
- **Early stopping (parado temprano):** Detener el proceso de aprendizaje tempranamente con algún criterio razonable.
- **Aumento de datos:** Generar nuevas muestras a partir de las muestras con las que contamos.
- **Auto-encoders:** Agregar una etapa no supervisada.

# Tipos de regularizaciones (algunos)

## ¿Cómo mejorar la capacidad de generalización del algoritmo?

- **Término de penalización (o regularizador):** Incluir un término adicional en la función costo.
- **Inyección de ruido:** Incluir algún tipo de ruido en el proceso de aprendizaje.
- **Early stopping (parado temprano):** Detener el proceso de aprendizaje tempranamente con algún criterio razonable.
- **Aumento de datos:** Generar nuevas muestras a partir de las muestras con las que contamos.
- **Auto-encoders:** Agregar una etapa no supervisada.

**Para la regularización es clave el conjunto de validación!**

# Penalización

Término de penalización que perturba la optimización de la función costo:

$$J(\theta) = \frac{1}{B} \sum_{i=1}^B L_i(\theta) + \lambda R(\theta)$$

## Penalización

Término de penalización que perturba la optimización de la función costo:

$$J(\theta) = \frac{1}{B} \sum_{i=1}^B L_i(\theta) + \lambda R(\theta)$$

### Motivación: Error de generalización

El regularizador trata ser representativo del error de generalización:

$$\mathbb{E}[L(\theta)] = \frac{1}{n} \sum_{i=1}^n L_i(\theta) + \left( \mathbb{E}[L(\theta)] - \frac{1}{n} \sum_{i=1}^n L_i(\theta) \right)$$

# Penalización

Término de penalización que perturba la optimización de la función costo:

$$J(\theta) = \frac{1}{B} \sum_{i=1}^B L_i(\theta) + \lambda R(\theta)$$

## Motivación: Error de generalización

El regularizador trata ser representativo del error de generalización:

$$\mathbb{E}[L(\theta)] = \frac{1}{n} \sum_{i=1}^n L_i(\theta) + \left( \mathbb{E}[L(\theta)] - \frac{1}{n} \sum_{i=1}^n L_i(\theta) \right)$$

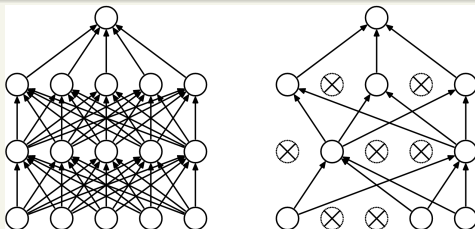
## Weight decay or L2 regularization

$$J(\theta) = \frac{1}{B} \sum_{i=1}^B L_i(\theta) + \sum_{l=1}^L \frac{\lambda_l}{2} \cdot \|\mathbf{w}^{[l]}\|_F^2 \rightarrow \frac{1}{2} \frac{\partial \|\mathbf{w}^{[l]}\|_F^2}{\partial \mathbf{w}^{[l]}} = \mathbf{w}^{[l]}$$

# Penalización en deep Learning: Weight decay

## Interpretación 1: Apagar neuronas

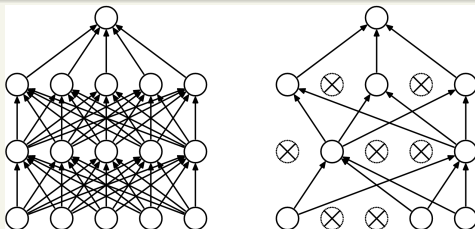
$w_{i,j}^{[l]} \approx 0$  simplifica la complejidad del modelo.



# Penalización en deep Learning: Weight decay

## Interpretación 1: Apagar neuronas

$w_{i,j}^{[l]} \approx 0$  simplifica la complejidad del modelo.



## Interpretación 2: Disminuir el máximo valor de la función costo

$$\mathbb{E}[L(\theta)] - \frac{1}{n} \sum_{i=1}^n L_i(\theta) \leq \max_{\phi \in \Theta} L(\phi)$$

Evita explosiones!

# Inyección de ruido

Se agrega ruido **ÚNICAMENTE** durante el entrenamiento.

## Tipos de inyección de ruido

- En los parámetros (no tan habitual)
- En las entradas (relacionado con data-augmentation)
- **En las unidades ocultas**

---

*Srivastava et al. 2014:* "Dropout: A simple way to prevent neural networks from overfitting".



# Inyección de ruido

Se agrega ruido **ÚNICAMENTE** durante el entrenamiento.

## Tipos de inyección de ruido

- En los parámetros (no tan habitual)
- En las entradas (relacionado con data-augmentation)
- **En las unidades ocultas**

## Dropout (inverted dropout technique)

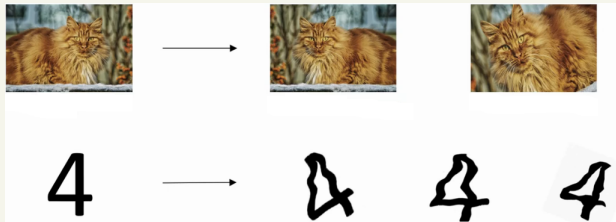
$$\mathbf{z}^{[l]} = \mathbf{W}^{[l]} \left( \mathbf{h}^{[l-1]} \odot \frac{1}{p_{\text{keep}}^{[l-1]}} \boldsymbol{\eta}^{[l-1]} \right) + \mathbf{b}^{[l]}, \quad \text{con } \boldsymbol{\eta}^{[l-1]} \sim \mathcal{B} \left( p_{\text{keep}}^{[l-1]} \right)$$

---

*Srivastava et al. 2014:* "Dropout: A simple way to prevent neural networks from overfitting".

# Data augmentation

Generar nuevas muestras inyectándole ruido a las que tenemos (noise injection a la entrada). Normalmente se incrementa la cantidad de muestras (no se reemplazan unas por otras). Muy poderoso cuando se usa información adicional sobre como pueden variar las muestras.



# Data augmentation

Generar nuevas muestras inyectándole ruido a las que tenemos (noise injection a la entrada). Normalmente se incrementa la cantidad de muestras (no se reemplazan unas por otras). Muy poderoso cuando se usa información adicional sobre como pueden variar las muestras.



4



4

4

4

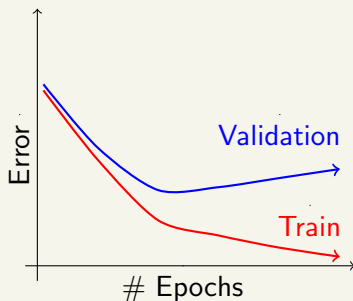
4



- Se incrementa la cantidad de muestras.
- Las nuevas muestras son más robustas a perturbaciones.
- Se pierde la independencia entre muestras.

# Early stopping

Para de entrenar antes!



- **PRO:** Para en el punto óptimo para la validación.
- **CONTRA:** Impacta directamente en el bias.

# Early stopping: Hiperparámetros más importantes

- **monitor**: Que métrica usamos para parar: “val\_loss”, “val\_accuracy”, “loss” o “accuracy”.
- **min\_delta**: Mínima variación que se considera una mejora (por debajo de esta no se considera mejora).
- **patience**: Cantidad de epochs que es aceptable no ver mejoras.
- **restore\_best\_weights**: En lugar de quedarte con el último modelo se queda con el mejor.

## Configuración por default:

monitor=val\_loss, min\_delta=0, patience=0, restore\_best\_weights=False

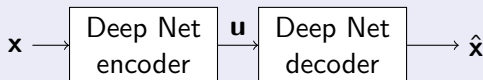
## Configuración para entrenar hasta convergencia:

monitor=loss, min\_delta=0, patience=20

# Autoencoder

Método de aprendizaje no supervisado que busca generar representaciones (típicamente de menor dimensión) al estilo PCA.

## Diagrama en bloques

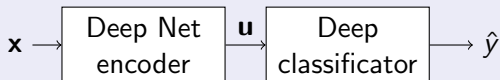


- La función costo más habitual es el error cuadrático medio  $\mathbb{E} \left[ \|\mathbf{X} - \hat{\mathbf{X}}\|_2^2 \right]$ .
- En un principio, las redes encoder y la decoder debían ser “espejadas”, hoy por hoy esto no es obligatorio.

# Autoencoder como regularización para la clasificación

## Preprocessing: Opción 1

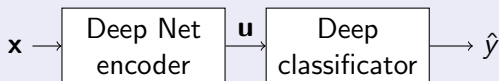
Entrenar el autoencoder y luego usar los  $\mathbf{u}$  para entrenar el clasificador.



# Autoencoder como regularización para la clasificación

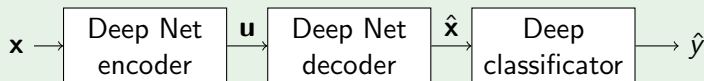
## Preprocessing: Opción 1

Entrenar el autoencoder y luego usar los  $\mathbf{u}$  para entrenar el clasificador.



## Preprocessing: Opción 2

Entrenar el autoencoder y luego usar los  $\hat{\mathbf{x}}$  para entrenar el clasificador.





# Autoencoder como regularización para la clasificación

## Regularizador: Opción 3

Entrenar simultáneamente ponderando las funciones costo

$$J(\theta) = \beta J_{\text{clas}}(\theta) + (1 - \beta) J_{\text{ae}}(\theta)$$

