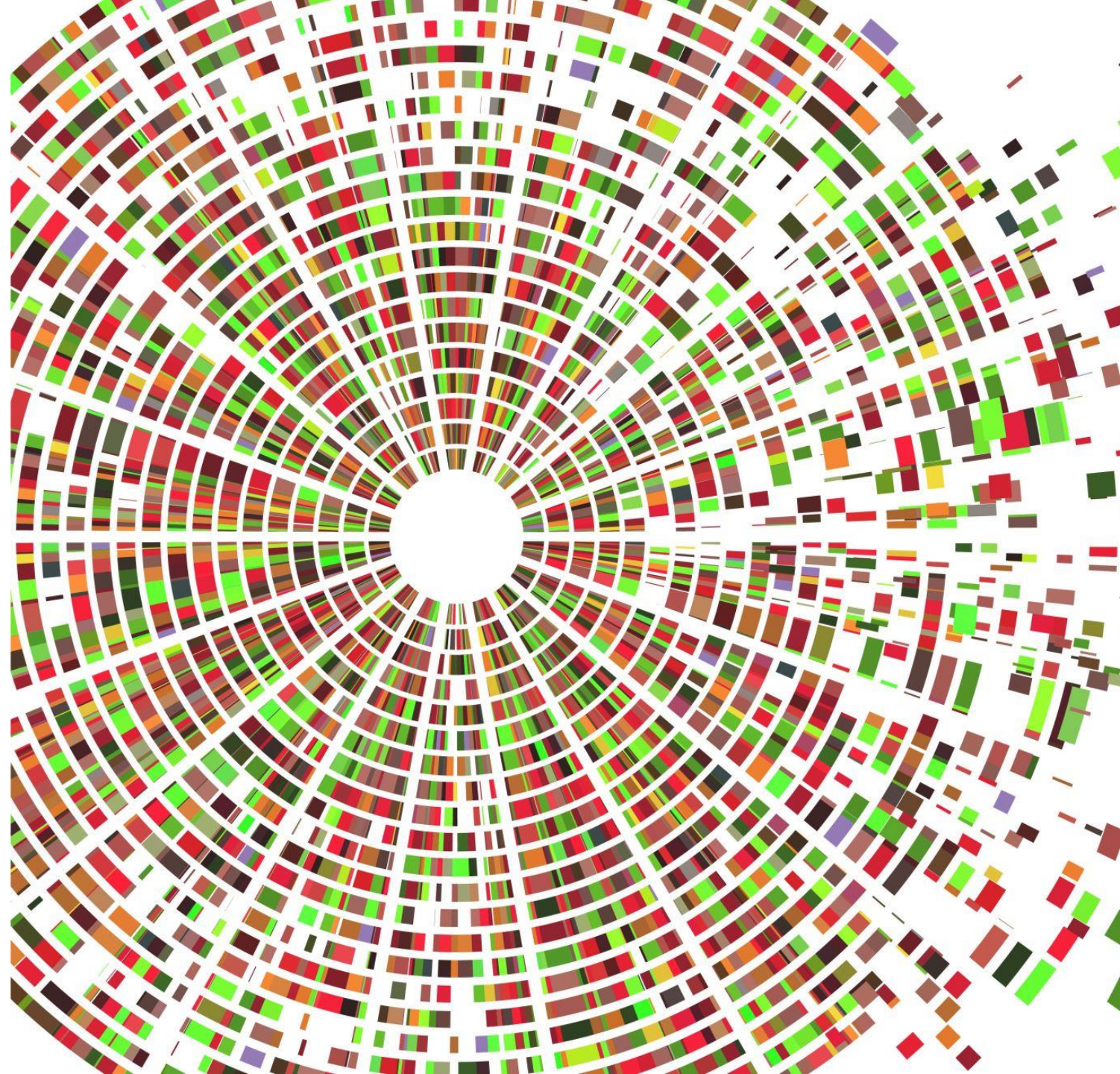


Métodos avanzados de ciencia de datos

Prof. Emily Díaz



Contenido



Continuación
de métodos
de
optimización



Métricas de
rendimiento



Sobreajuste y
balance entre
sesgo y varianza



Métodos de
regularización

Otros métodos de optimización de pesos



Otros algoritmos de optimización

Descendiente de gradiente

- Es el más básico.
- Es un método de optimización de primer orden (primera derivada)
- Es fácil de calcular, implementar y entender
- Desventajas: Puede quedarse atascado en mínimo local y requiere mucha memoria para calcular los gradientes de un set de datos completo (SGD solventa en cierta parte estos problemas pero añadir alta varianza a los parámetros)

Otros algoritmos de optimización

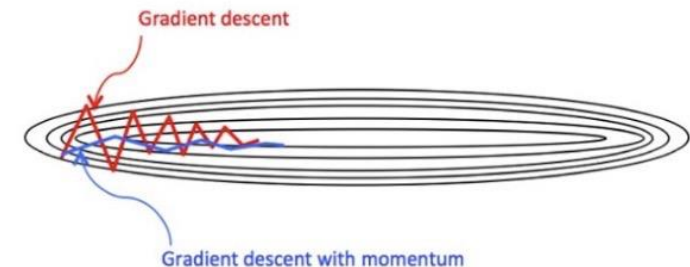
Momentum

- Se inventó para reducir la alta varianza en SGD y suaviza la convergencia. Acelera la convergencia hacia la dirección relevante y reduce la fluctuación hacia la dirección irrelevante.
- Se introduce otro hiperparámetro conocido como momentum, simbolizado por " ρ ".
- Su concepto es similar a una pelota que rueda cuesta abajo, acumulando velocidad y suavizando así el proceso de optimización. Si esta pelota encuentra un obstáculo en su camino, como un hoyo o un terreno plano sin pendiente, su velocidad acumulada v le daría a la pelota la fuerza suficiente para rodar sobre este obstáculo.
- Desventajas: Añade otro hiperparámetro que ajustar

$$v_{t+1} \leftarrow \rho v_t + \nabla_{\theta} \mathcal{L}(\theta)$$

$$\theta_j \leftarrow \theta_j - \epsilon v_{t+1}$$

Donde v es "velocidad", ρ es el factor de decaimiento exponencial entre 0 y 1 que determina la contribución relativa del gradiente actual y los gradientes anteriores al cambio de peso, ϵ es la tasa de aprendizaje y θ es el peso (Ws)



SGD con momentum – pseudo código

Require: Learning rate ϵ , momentum parameter α .

[Goodfellow, 2016]

Require: Initial parameter θ , initial velocity v .

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(m)}\}$ with corresponding targets $\mathbf{y}^{(i)}$.

Compute gradient estimate: $\mathbf{g} \leftarrow \frac{1}{m} \nabla_{\theta} \sum_i L(f(\mathbf{x}^{(i)}; \theta), \mathbf{y}^{(i)})$

Compute velocity update: $\mathbf{v} \leftarrow \alpha \mathbf{v} - \epsilon \mathbf{g}$

Apply update: $\theta \leftarrow \theta + \mathbf{v}$

end while

* Cambio de nomenclatura a como estaba antes,
 $\alpha = \rho$

Otros algoritmos asociados a problema de tasa de aprendizaje



En los algoritmos anteriores, la tasa de aprendizaje es constante para todos los parámetros y durante todos los periodos de entrenamiento.



Esto, puede ser un problema porque cerca al punto óptimo, queremos que vaya más lento y lejos que cambie con mayor velocidad. Además, la magnitud de los gradientes puede variar significativamente entre sí por lo que pueden requerir distintos ritmos.

1. Adagrad

- Ajusta la tasa de aprendizaje **individualmente** para cada parámetro, escalándola inversamente con la **raíz cuadrada de la suma de todos los gradientes cuadrados históricos**.
- Esto permite actualizaciones más grandes para características que ocurren con poca frecuencia y actualizaciones más pequeñas para características frecuentes.
- Sin embargo, una desventaja es que la tasa de aprendizaje puede volverse excesivamente pequeña con el tiempo.

AdaGrad

$$g_0 = 0$$

$$g_{t+1} \leftarrow g_t + \nabla_{\theta} \mathcal{L}(\theta)^2 \longrightarrow$$

$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_{\theta} \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

Acumulación de la suma de gradientes al cuadrado de todos los periodos que han pasado

2. RMSprop (Root Mean Square Propagation)

- Es una mejora con respecto a AdaGrad, que introduce un factor de decaimiento (α) en la suma de los gradientes al cuadrado.
- Esto evita que la tasa de aprendizaje sea demasiado pequeña, lo que soluciona el problema de la tasa de aprendizaje decreciente en AdaGrad.

RMS Prop

$$g_0 = 0, \alpha \simeq 0.9$$

$$g_{t+1} \leftarrow \alpha \cdot g_t + (1 - \alpha) \nabla_{\theta} \mathcal{L}(\theta)^2$$

$$\theta_j \leftarrow \theta_j - \epsilon \frac{\nabla_{\theta} \mathcal{L}}{\sqrt{g_{t+1}} + 1e^{-5}}$$

3. Adam (Adaptive Moment Estimation)

- Combina las ideas de **AdaGrad** y **RMSprop** manteniendo un promedio decreciente exponencial de gradientes pasados (*momentum*) y gradientes al cuadrado.
- Esto permite a Adam adaptar la tasa de aprendizaje en función de los gradientes pasados y, al mismo tiempo, tener en cuenta el *momentum*, lo que genera un entrenamiento más eficiente y sólido.
- Usa el primer (media) y segundo momento (varianza no centrada) de los gradientes.

	$m_0 = 0, v_0 = 0$	
Primer momento	$m_{t+1} \leftarrow \beta_1 m_t + (1 - \beta_1) \nabla_{\theta} \mathcal{L}(\theta)$	Similar a SGD + momentum excepto el $(1-\beta)$
Segundo momento	$v_{t+1} \leftarrow \beta_2 v_t + (1 - \beta_2) \nabla_{\theta} \mathcal{L}(\theta)^2$	Similar a RMSProp
	$\theta_j \leftarrow \theta_j - \frac{\epsilon}{\sqrt{v_{t+1}} + 1e^{-5}} m_{t+1}$	Combinación de ambos



Métricas de rendimiento



Métricas de rendimiento: ¿Cómo saber si nuestra red neuronal tiene buen rendimiento o no?

Clásicas de clasificación

- Matriz de confusión:
Exactitud,
Precisión,
Recall, *F1 score*
- ROC-AUC

Clásicas de regression

- MSE
- MAE
- MAPE
- R cuadrado y r cuadrado ajustado

Específicas para redes neuronales

- Monitoreo de métricas por epoch y set de datos

Fórmulas de regresión

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

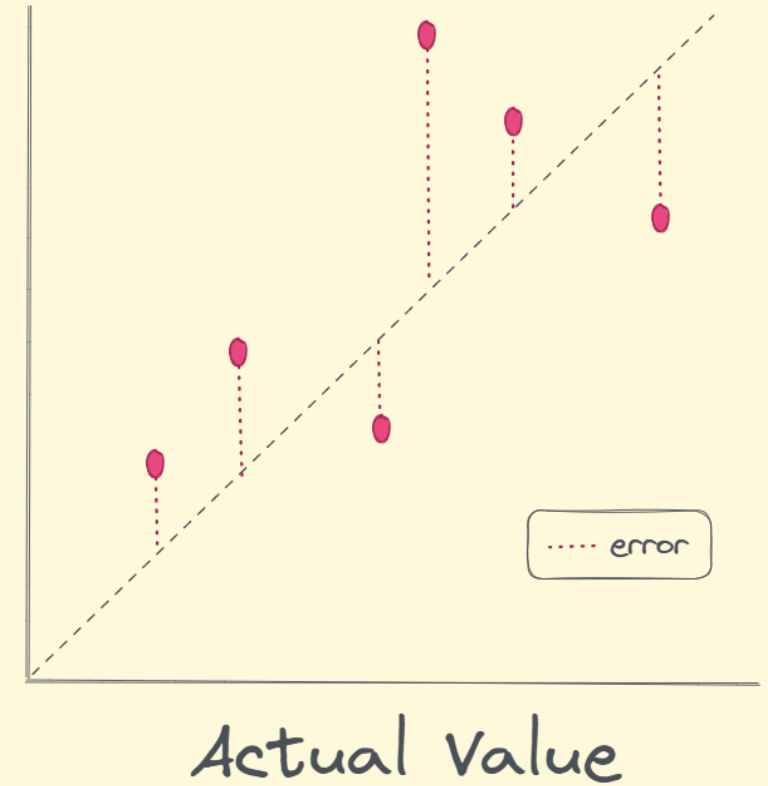
$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}$$

$$R^2 = 1 - \frac{SSR}{SST} = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

Predicted Value



Fórmulas de clasificación

	Positive — Predicted — Negative	
Negative — Actual — Positive	<div>True Positive (TP) Correct objects</div>	<div>False Negative (FN) Missed objects</div>
	<div>False Positive (FP) Extra objects</div>	<div>True Negative (TN) No objects</div>
	<div>Precision $\frac{TP}{(TP+FP)}$</div>	<div>Negative predictive value $\frac{TN}{(TN+FP)}$</div>

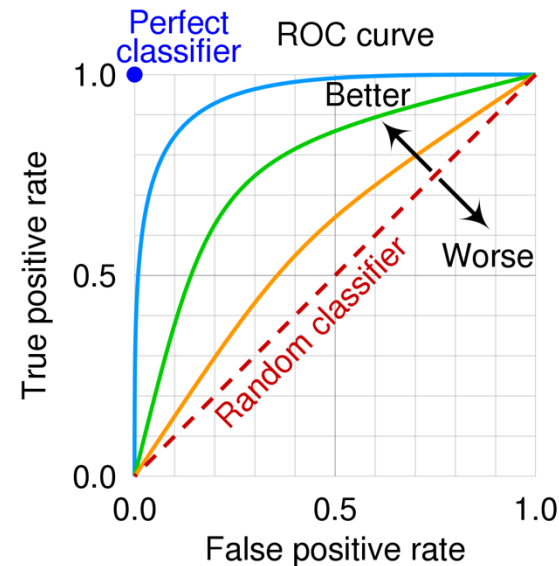
$$\text{Recall} = \frac{TP}{(TP+FN)}$$

$$\text{Accuracy} = \frac{(TP+TN)}{(TP+TN+FP+FN)}$$

$$\text{Specificity} = \frac{TN}{(TN+FP)}$$

$$F1 = \frac{2 \times \text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$$

Ideal cuando se tiene desbalance de clases



$$\text{True Positive Rate (TPR)} = \frac{TP}{P} = \frac{TP}{TP + FN}$$

also called sensitivity/recall/hit rate

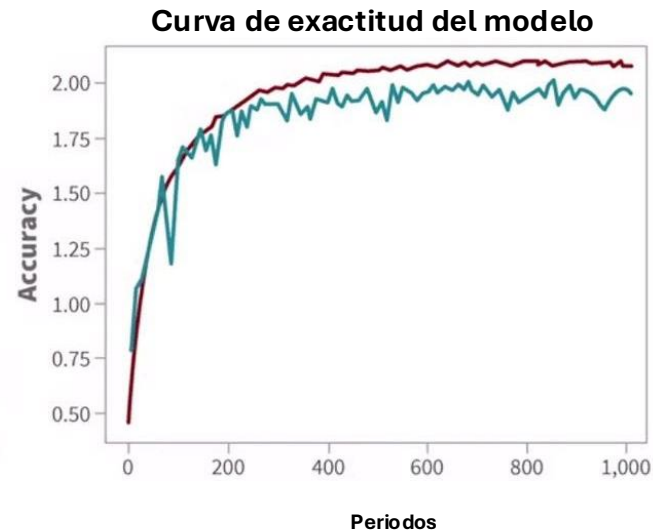
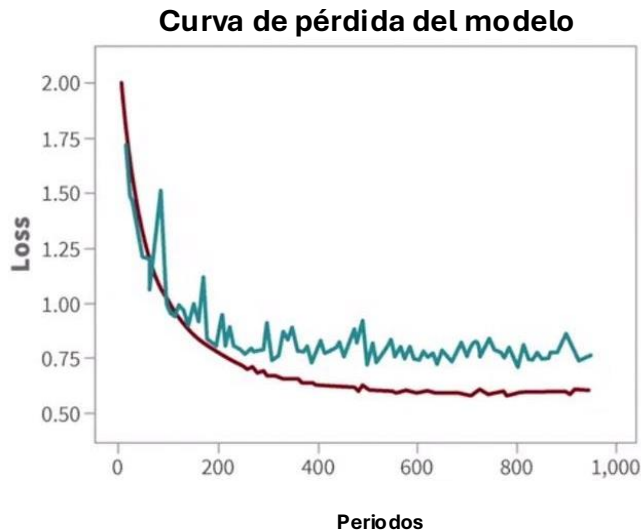
$$\text{False Positive Rate (FPR)} = \frac{FP}{N} = \frac{FP}{FP + TN}$$

also called fall out

Area bajo la curva (AUC ROC): Entre 0 y 1, valores más altos son los de mejor rendimiento
No necesita un punto de corte, no es sensible a desbalance

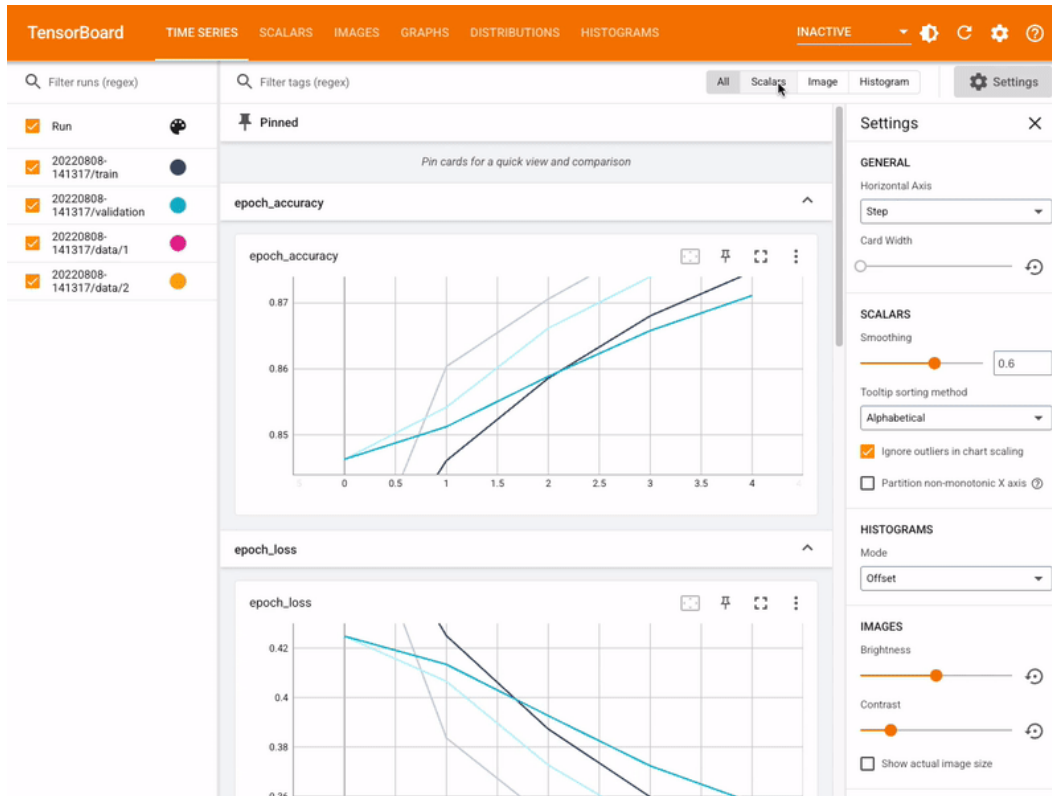
Monitoreo de rendimiento

— Entrenamiento
— Validación



- **Tendencia:** Esperamos que conforme pasan iteraciones, el modelo vaya mejorando
- **Estabilidad:** Que no haya mucha fluctuación en los resultados (bajas y altas muy pronunciadas)
- **Relación entrenamiento/validación:** se espera que la validación tenga más alto error pero queremos que la brecha no sea muy amplia ni se vaya incrementando en el tiempo (indicador de sobreajuste)
- **Convergencia:** Al final del entrenamiento, las curvas de entrenamiento y validación deben estabilizarse en un valor, indica que se ha aprendido suficiente

Tensorboard – Visualización del desempeño



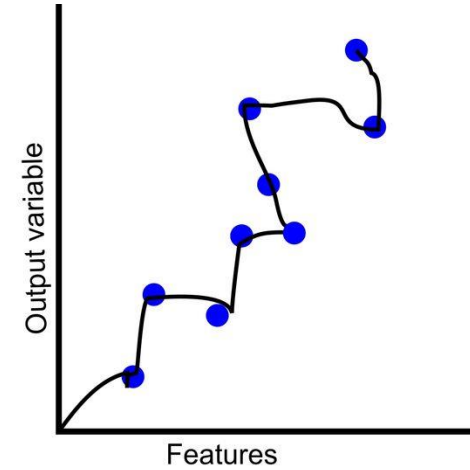
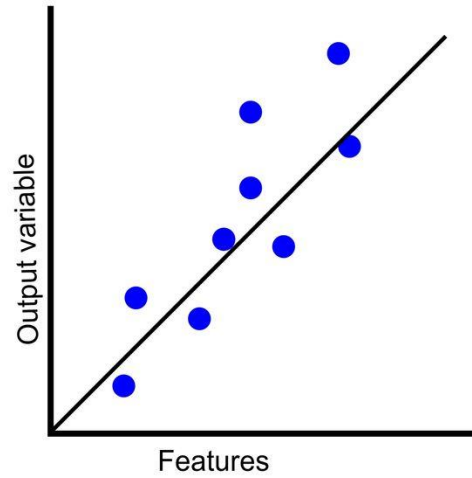
TensorBoard es una poderosa herramienta de visualización proporcionada por TensorFlow que ayuda a monitorear y depurar el rendimiento de las redes neuronales durante el entrenamiento y la evaluación. Algunos de sus beneficios son:

- Seguimiento de los experimentos en tiempo real
- Puede comparar distintas corridas
- Fácil de compartir los resultados con otros
- Se pueden visualizar métricas más específicas como tasas de aprendizaje, uso de memoria y otras.

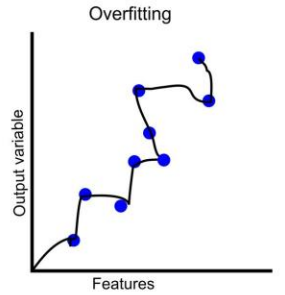
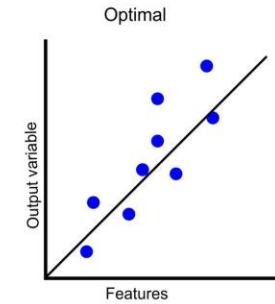
• Sesgo, varianza y sobreajuste



¿Qué es sobreajuste?



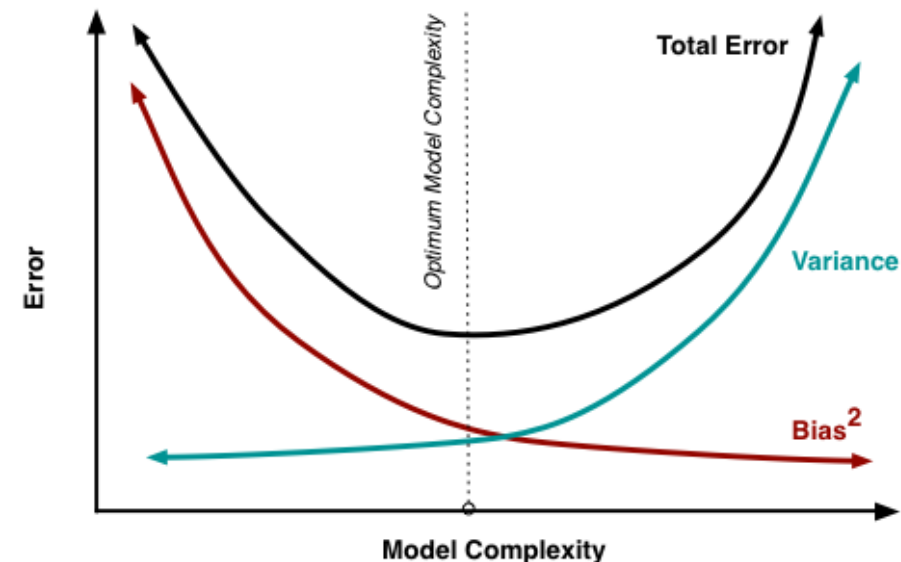
¿Qué es sobreajuste?



- Se produce cuando un modelo da predicciones altas de entrenamiento pero no para datos nuevos (no vistos por el modelo a la hora de entrenarse)
- Un modelo con sobreajuste es un modelo no ideal pues no extrapola a otros datos, no aprendió a generalizar
- Cuándo sucede esto:
 - Muy pocos datos que no logran representar bien todos los valores posibles
 - Se entrena un modelo por mucho tiempo en los mismos datos
 - La complejidad del modelo es muy alta
- Cuáles son formas de detectarlo:
 - Comparar rendimiento en datos de entrenamiento con los de testeo

Sesgo y varianza, su conexión a sobre-ajuste y falta de ajuste

- Concepto relacionado al rendimiento del modelo y su generalización
- Equilibrio entre sesgo y varianza: relación entre complejidad del modelo y el rendimiento
- Son parte de la descomposición del error esperado: error irreducible, sesgo y varianza.
- Sesgo (bias): Monto por el cual el promedio de los estimados difieren de la media real (diferencia entre el valor esperado y el real). Es el error dado los supuestos del modelo
- Varianza: La diferencia entre el promedio de las predicciones al cuadrado y el cuadrado de la predicción promedio. Es el error dada la sensibilidad del modelo a fluctuaciones en los datos de entrenamiento



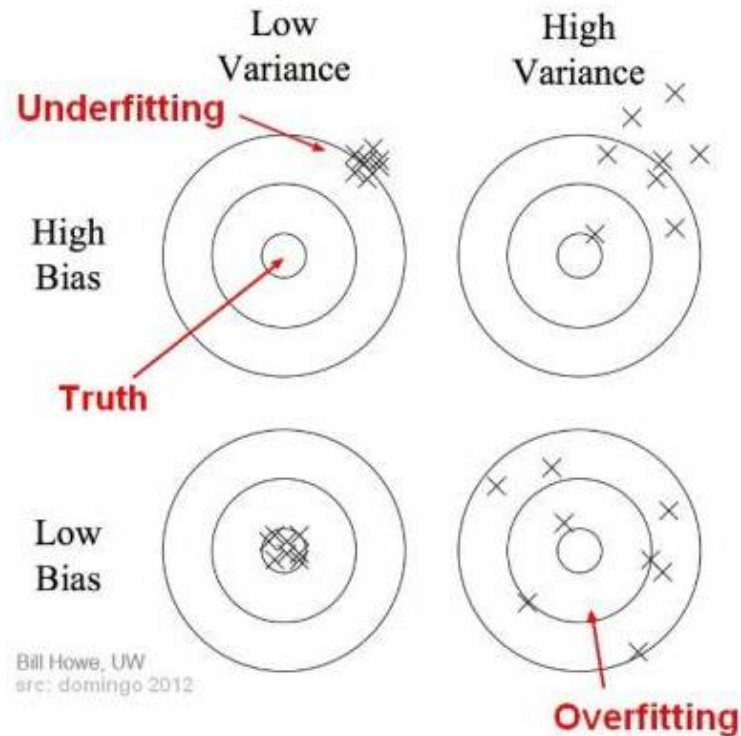
$$\begin{aligned}\text{Err}(x_0) &= E[(Y - \hat{f}(x_0))^2 | X = x_0] \\ &= \sigma_\epsilon^2 + [E\hat{f}(x_0) - f(x_0)]^2 + E[\hat{f}(x_0) - E\hat{f}(x_0)]^2 \\ &= \sigma_\epsilon^2 + \text{Bias}^2(\hat{f}(x_0)) + \text{Var}(\hat{f}(x_0)) \\ &= \text{Irreducible Error} + \text{Bias}^2 + \text{Variance}.\end{aligned}$$

Verdadera función subyacente

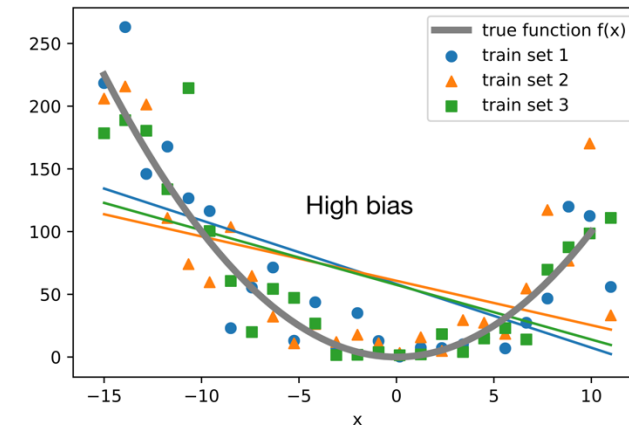
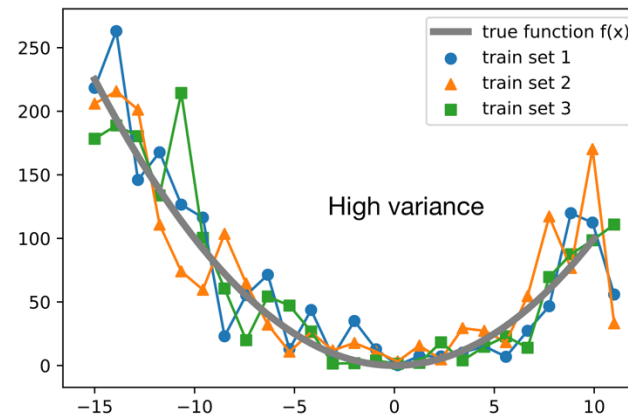
Predicción del modelo para x_0

Predicción esperada sobre todos los posibles set de datos de entrenamiento

Relación con sobreajuste y falta de ajuste



- El equilibrio ideal entre sesgo y varianza se logra cuando un modelo tiene el nivel justo de complejidad para capturar con precisión los patrones subyacentes en los datos sin sobreajustarse al ruido.
- Lo ideal es elegir un modelo que **capture con precisión las regularidades** de sus datos de entrenamiento, pero que también **se generalice bien** a datos no vistos.
- Desafortunadamente, normalmente es imposible hacer ambas cosas simultáneamente. Los métodos de aprendizaje de **alta varianza** pueden ser capaces de representar bien su conjunto de entrenamiento, pero corren el riesgo de **sobreajustarse** a datos de entrenamiento ruidosos o no representativos. Por el contrario, los algoritmos con **alto sesgo** suelen producir modelos más simples que pueden no capturar regularidades importantes (es decir, **no ajustarse lo suficiente**) en los datos.



Underfitting/falta de adaptación

- Se produce cuando un modelo es muy sencillo y no logra predecir con exactitud los datos
- Poseen alto sesgo y baja varianza
- Cuándo sucede esto:
 - Se entrena un modelo por poco tiempo
 - El modelo es muy simple para el problema
- Cómo detectarlo:
 - Mal desempeño tanto en los datos de entrenamiento como los de testeo


Como remediarlos

- Si se tiene underfitting/falta de ajuste, se pueden intentar estas modificaciones:
 - Añadir variables
 - Utilizar un modelo más complejo
 - Entrenar por más periodos
- Si se tiene overfitting/sobre-ajuste se puede intentar:
 - Uso de modelo menos complejo
 - Uso de más datos (data augmentation o más muestras reales)
 - Regularización





Métodos de regularización



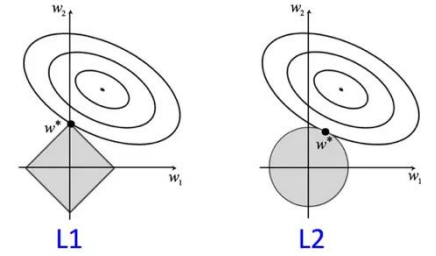


Métodos de regularización para redes neuronales



- Son formas de prevenir el sobreajuste
 - Hay métodos que se utilizan para regresión lineal que se utilizan también para redes neuronales:
 1. Penalización de grandes coeficientes: L1 y L2
 2. Dropout/abandon o dilución
 3. Early stopping/finalización temprana
- 

L1 y L2



L1 (Lasso)

- Agrega una penalización igual al **valor absoluto** de la magnitud de los coeficientes.
- Esto hace que algunos coeficientes se vuelva cero, convirtiéndolo en un método de selección de variables

Función de pérdida

$$= \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M |W_j|$$

L2 (Ridge)

- Agrega una penalización igual al cuadrado de la magnitud de los coeficientes
- Esto tiende a reducir los coeficientes de manera uniforme, pero no los elimina.

Función de pérdida

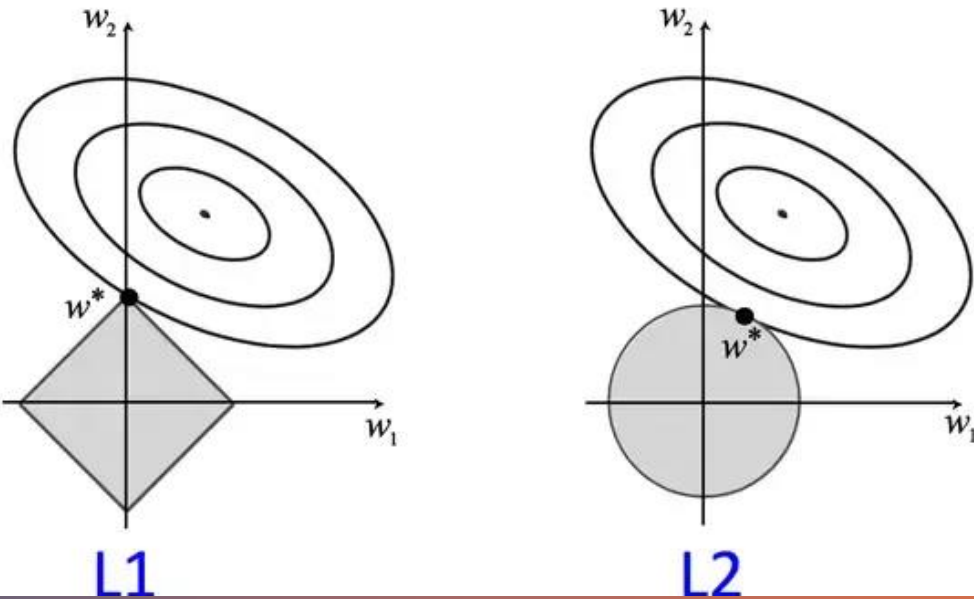
$$= \sum_{i=0}^N (y_i - \sum_{j=0}^M x_{ij} W_j)^2 + \lambda \sum_{j=0}^M W_j^2$$

L2 Regularization (Ridge):

Adds a penalty equal to the square of the coefficients' magnitude.

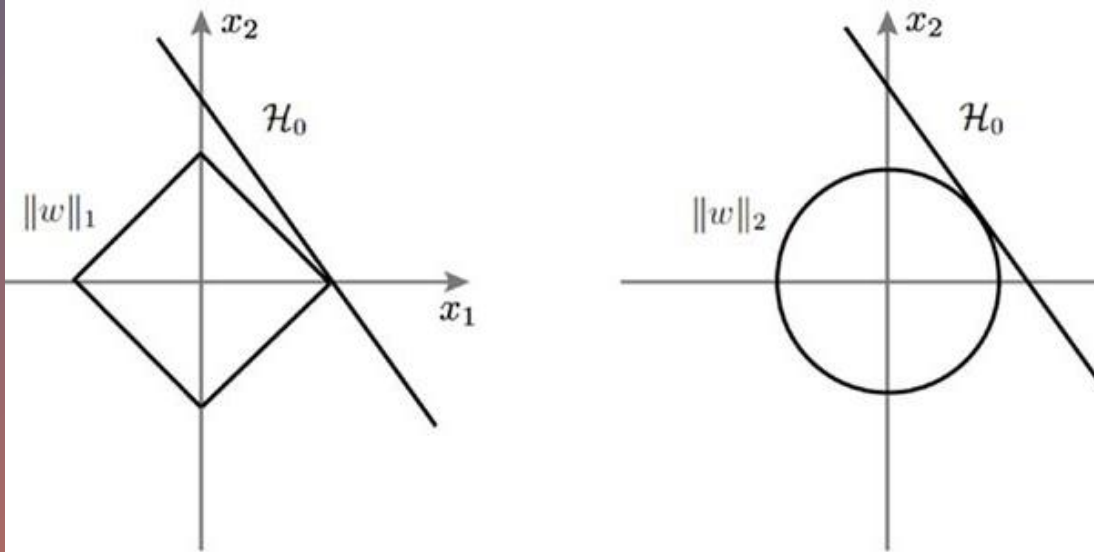
This tends to shrink coefficients uniformly but doesn't eliminate them.

L1 y L2

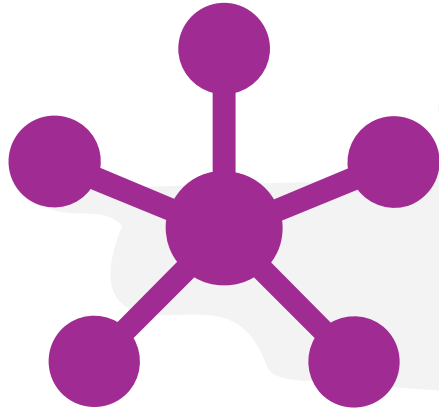


L1 regularization

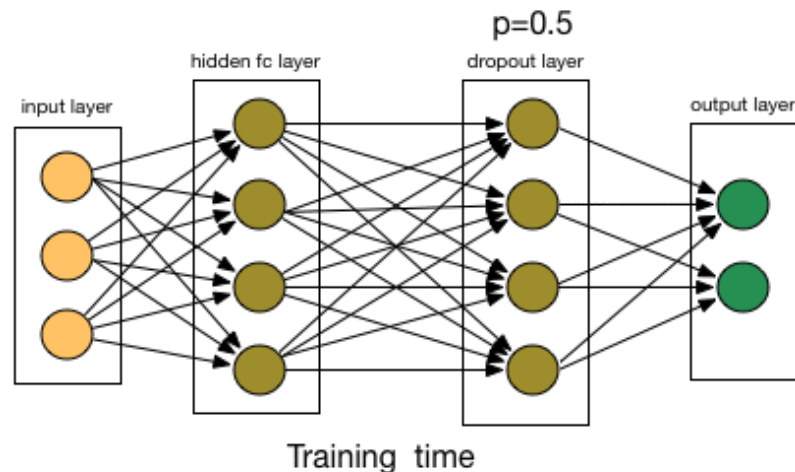
B L2 regularization



- El concepto de círculo unitario en 2D (el conjunto de puntos que tienen norma igual a 1) está asociado a las características de la regularización de L1 y L2
- Estamos tratando de minimizar una función de costo que incluye la penalización de los coeficientes. El término de regularización penaliza las soluciones que se alejan mucho del origen (norma alta), lo que obliga al modelo a elegir coeficientes que estén "cerca" de este espacio de la norma.
- Por la forma de cada una de las normas utilizadas es que en el caso de L1 funciona como selección de variables y en L2 como repartición de la reducción de coeficientes pero no eliminación



Dropout/dilución



- Es una técnica de regularización específica para redes neuronales artificiales. Realiza una eliminación aleatoria de neuronas (ocultas como visibles) durante el entrenamiento
- Por qué esto se considera regularización? Obliga a la red a no depender demasiado de ninguna neurona individual.
- Introduce un hiperparámetro: probabilidad de eliminación o tasa de abandono
- Equivale a entrenar muchas redes pequeñas independientes que se ensamblan para obtener el resultado final.
- Los valores finales de los pesos no se agregan explícitamente, sino que son el resultado del proceso de entrenamiento en el que la red aprende a optimizar los pesos bajo la restricción de abandono. El abandono esencialmente hace que la red sea más resistente y menos dependiente de pesos específicos, lo que genera valores de peso finales más generalizados y robustos.
- Valores comunes del parámetro $p = 0.2, 0.3, 0.5$

Dropout/dilución

- Proceso:
 - En cada propagación hacia delante, define el valor de salida de cada neurona como cero con probabilidad P . Las “ignoradas” no participan en esa propagación hacia delante ni retropropagación
 - Las neuronas restantes se escalan $\frac{1}{1-p}$ por durante el entrenamiento para mantener el valor esperado del resultado. Por ejemplo, si la tasa de abandono es 0.5, entonces la salida de las neuronas restantes se multiplica por 2 para compensar el 50 % de neuronas que se abandonaron.
 - Durante la inferencia, no se descarta ninguna neurona, sino que se utilizan todas, pero sus pesos se reducen según la tasa de abandono, que promedia efectivamente los resultados de las diferentes redes "adelgazadas" que se entrenaron.
 - A lo largo de múltiples iteraciones de entrenamiento, diferentes subconjuntos de neuronas están activos y los pesos se ajustan en consecuencia.
 - Los valores de peso finales representan una especie de promedio implícito de todas estas subredes, lo que da como resultado un conjunto de pesos que funcionan bien incluso cuando no se descartan neuronas durante la inferencia.



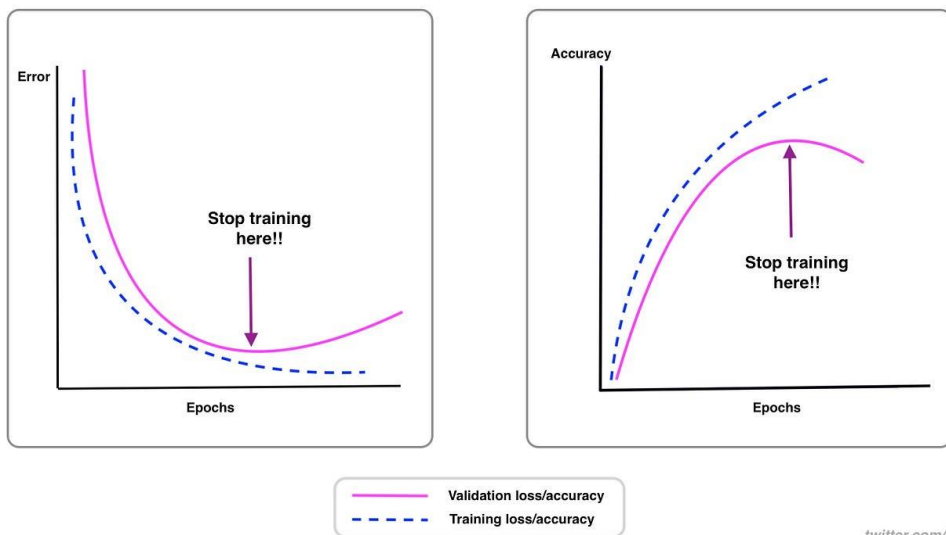


Dropout/dilución

- El posicionamiento de esta capa generalmente se aplica a capas completamente conectadas en redes neuronales, pero también se puede utilizar en otros tipos de capas, como capas convolucionales, aunque a menudo con una tasa de caída menor.
- **Cómo aplicarlo:**
 - Redes pequeñas: Se puede aplicar la eliminación a las capas más críticas (normalmente las capas ocultas) para garantizar que la red no se sobreajuste al principio.
 - Redes más profundas: en redes más profundas, la eliminación se puede aplicar a varias capas, pero normalmente no a todas. Aplicar la eliminación a todas las capas puede provocar un subajuste, especialmente en redes más profundas donde las distintas capas capturan distintos niveles de abstracción.
 - Por lo general, **no se aplica a la capa de salida**. La dilución en la capa de salida puede interferir con las predicciones finales, lo que genera inestabilidad.

Early stopping/detención temprana

Early Stopping



- Funciona dividiendo el conjunto de entrenamiento original en un nuevo conjunto de entrenamiento y un conjunto de validación. El error en el conjunto de validación se utiliza como proxy del error de generalización para determinar cuándo ha comenzado el sobreajuste.
- Proceso:
 - Se dividen los datos de entrenamiento en un conjunto de entrenamiento y un conjunto de validación
 - Se entrena solo en el conjunto de entrenamiento y se evalúa el error en el conjunto de validación cada period/epoch
 - Se detiene el entrenamiento cuando el error en validación tienda a incrementar (o cualquier otra métrica de desempeño monitoreada empieza a deteriorarse)
- Pero cuándo sabemos que la red está empeorando en rendimiento y no es parte de la estabilización?
 - Se añade un **hiperparámetro de paciencia** que controla cuántos periodos adicionales se permite entrenar luego de no ver mejoras en el rendimiento sobre el set de validación.

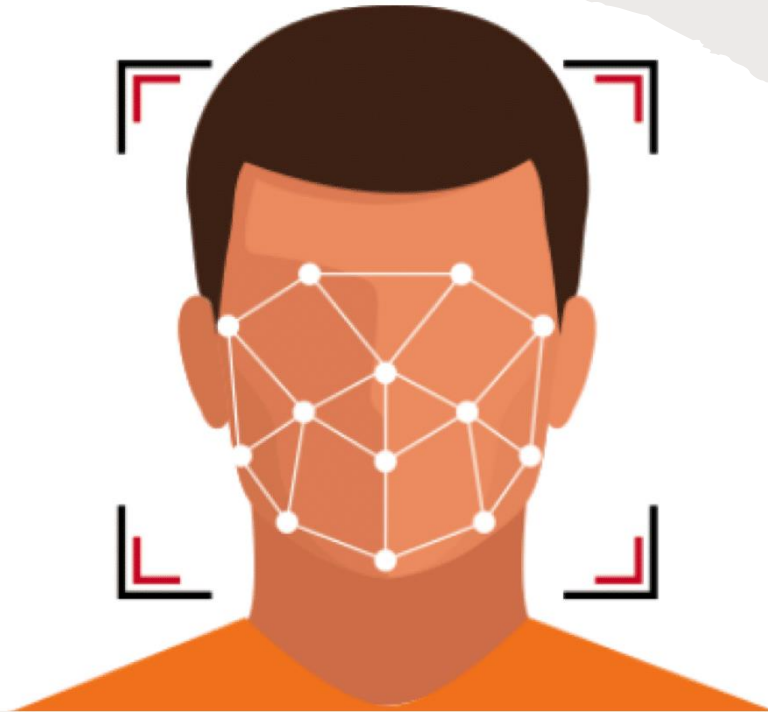


Ejemplo de métricas de rendimiento



Problema 1:

- Hemos construido una red neuronal para obtener la probabilidad de que una imagen contenga una cara. Para ello hemos utilizado miles de imágenes de caras y de no caras para entrenarlo. Ahora queremos evaluar el poder predictivo del modelo sobre las imágenes:



Observación 1



Observación 2



Observación 3



Observación 4

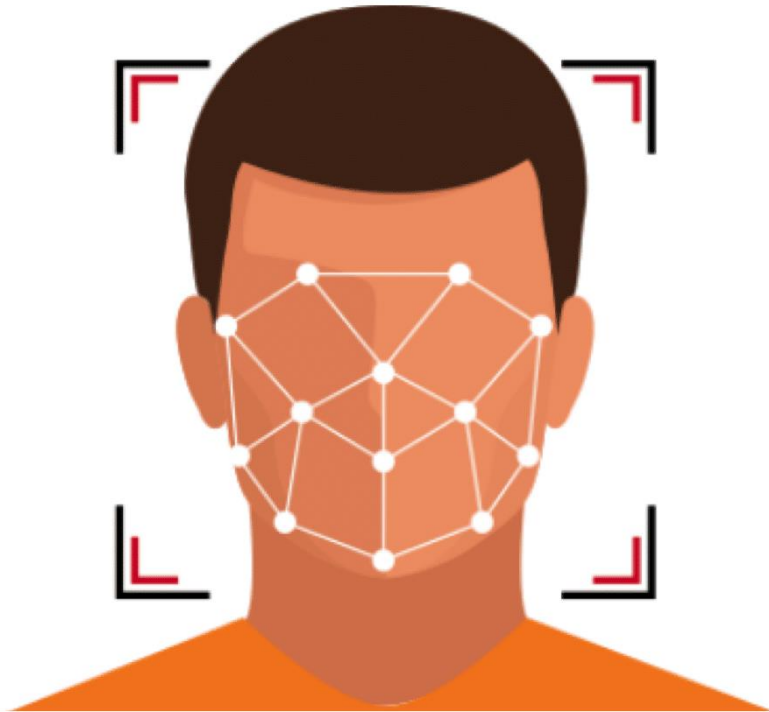


Observación 5



Problema 1:

- Al predecir con el modelo las cinco observaciones, nos devuelve las siguientes probabilidades de cada imagen, respectivamente: [0.9, 0.6, 0.7, 0.3, 0.1]
- Considerando un punto de corte de 0.5, calcula: % de aciertos, recall, precisión y f1



Observación 1



Observación 2



Observación 3



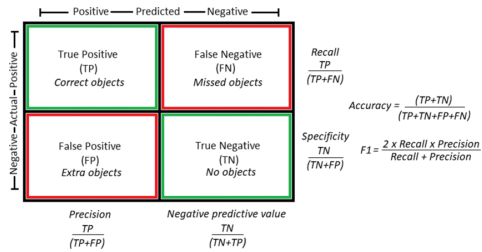
Observación 4



Observación 5



Problema 1:



		Predicho	
		0	1
Real	0	1	1
	1	1	2

% de aciertos = $3/5 = 0.6 = 60\%$

Precisión = $2/(2+1) = 2/3 = 67\%$

Recall = $2/(2+1) = 2/3 = 67\%$

F1 = $2 * (2/3 * 2/3) / (2/3 + 2/3) = 67\%$

- Al predecir con el modelo las cinco observaciones, nos devuelve las siguientes probabilidades de cada imagen, respectivamente: [0.9, 0.6, 0.7, 0.3, 0.1]
- Considerando un punto de corte de 0.5, calcula: % de aciertos, recall, precisión y f1

Observación 1



Observación 2



Observación 3



Observación 4



Observación 5



Problema 2



- Luego de que a la generación de estadística 2024 sacaran excelentes notas en el examen, deciden entrenan una red neuronal con la información de cada estudiante para que las generaciones siguientes puedan predecir su nota final de este curso.
- Para verificar si es un buen modelo, dejaron una muestra de 5 estudiantes para testeo. Con ello, deben calcular el MSE (error cuadrático medio) y MAPE (error porcentual absoluto medio). Las notas reales de los estudiantes en el set de test son: [100, 90, 85, 95, 90]
- El modelo predijo los siguientes valores: [92, 89, 85, 97, 100]

Problema 2

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

- Las notas reales de los estudiantes en el set de test son: [100, 90, 85, 95, 90]
- El modelo predijo los siguientes valores: [92, 89, 85, 97, 100]

Solución:

Errores = [100-92, 90-89, 85-85, 95-97, 90-100] = [8, 1, 0, -2, -10]

$$MSE = \frac{1}{5} * (8^2 + 1^2 + 0^2 + (-2)^2 + (-10)^2) = \frac{169}{5} = 33.8$$

$$MAPE = \frac{1}{5} * \left(\frac{8}{100} + \frac{1}{90} + 0 + \frac{2}{95} + \frac{10}{90} \right) \\ = \frac{1}{5} * 0.2233 = 4.47\%$$





Problema 3

- Hemos desarrollado una red neuronal que predice el tueste del café mediante características del sabor medidas por baristas de Costa Rica. Lo han dividido en tueste claro, medio y oscuro. Queremos medir el porcentaje de aciertos totales y por categoría que hemos tenido y para eso hemos dejado 10 muestras de café para testeo.
- Hemos codificado las clases de la siguiente manera: claro = 0, medio = 1 y oscuro = 2
- Las muestras son: [0,0,1,2,2,2,1,1,0,2]
- Las probabilidades resultantes del modelo son:

0.2	0.1	0.7
0.8	0.1	0.1
0.3	0.6	0.1
0.1	0.1	0.8
0.1	0.6	0.3
0.8	0.1	0.1
0.1	0.7	0.2
0.2	0.2	0.6
0.6	0.2	0.2
0.1	0.7	0.2



Problema 3

- Las muestras son: [0,0,1,2,2,2,1,1,0,2]
- Las probabilidades resultantes del modelo son:

0.2	0.1	0.7
0.8	0.1	0.1
0.3	0.6	0.1
0.1	0.1	0.8
0.1	0.6	0.3
0.8	0.1	0.1
0.1	0.7	0.2
0.2	0.3	0.5
0.4	0.3	0.3
0.1	0.7	0.2

→ [2,0,1,2,1,0,1,2,0,1]

Porcentaje de casos correctamente clasificados:

$$5/10 = 50\%$$

$$\text{Clase 0: } \frac{2}{3} = 67\%$$

$$\text{Clase 1: } \frac{2}{3} = 67\%$$

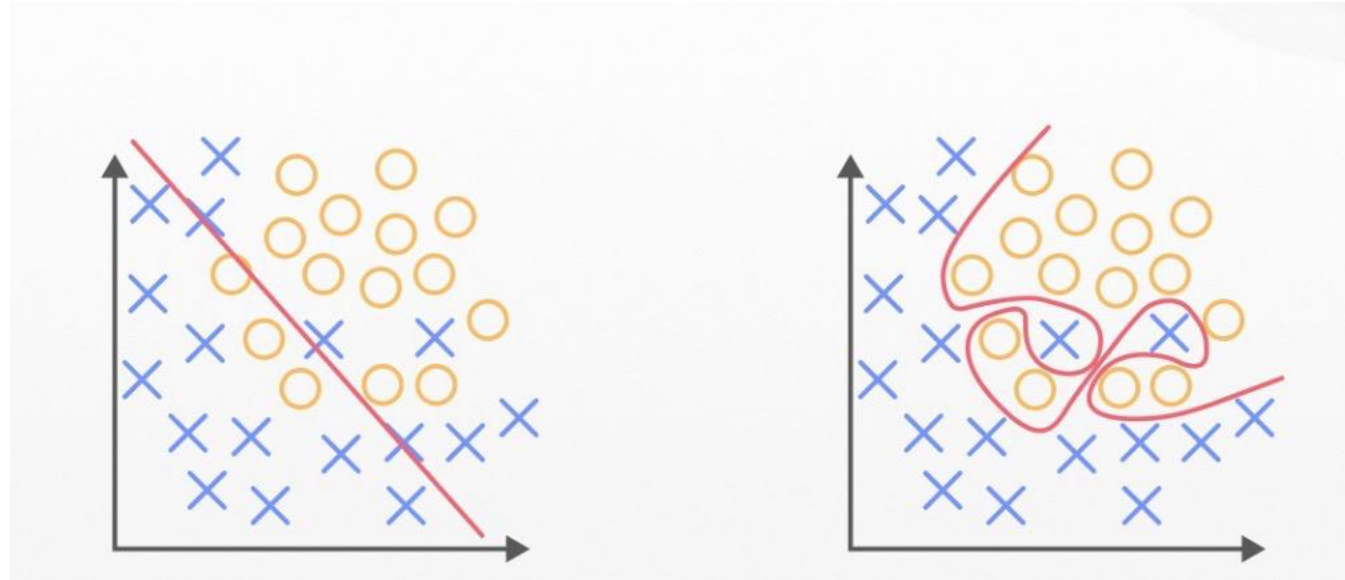
$$\text{Clase 2: } \frac{1}{4} = 25\%$$



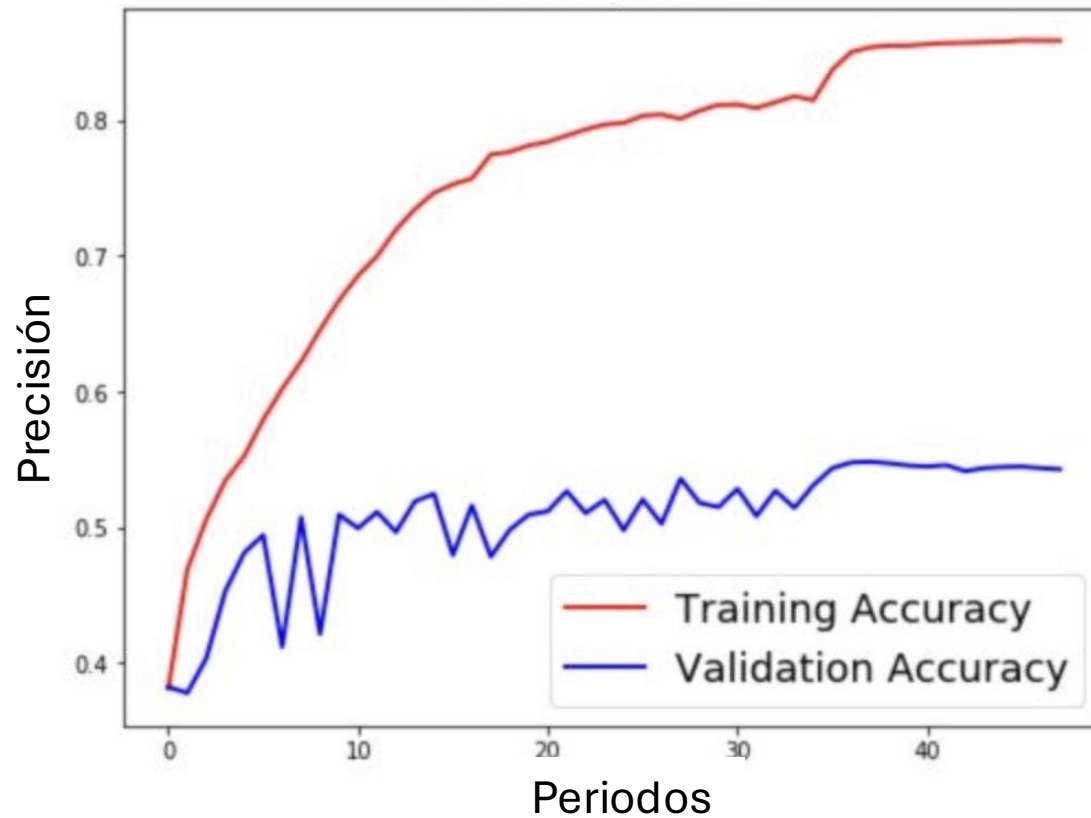
Ejemplo de sobreaajuste y falta de ajuste y sus remedios



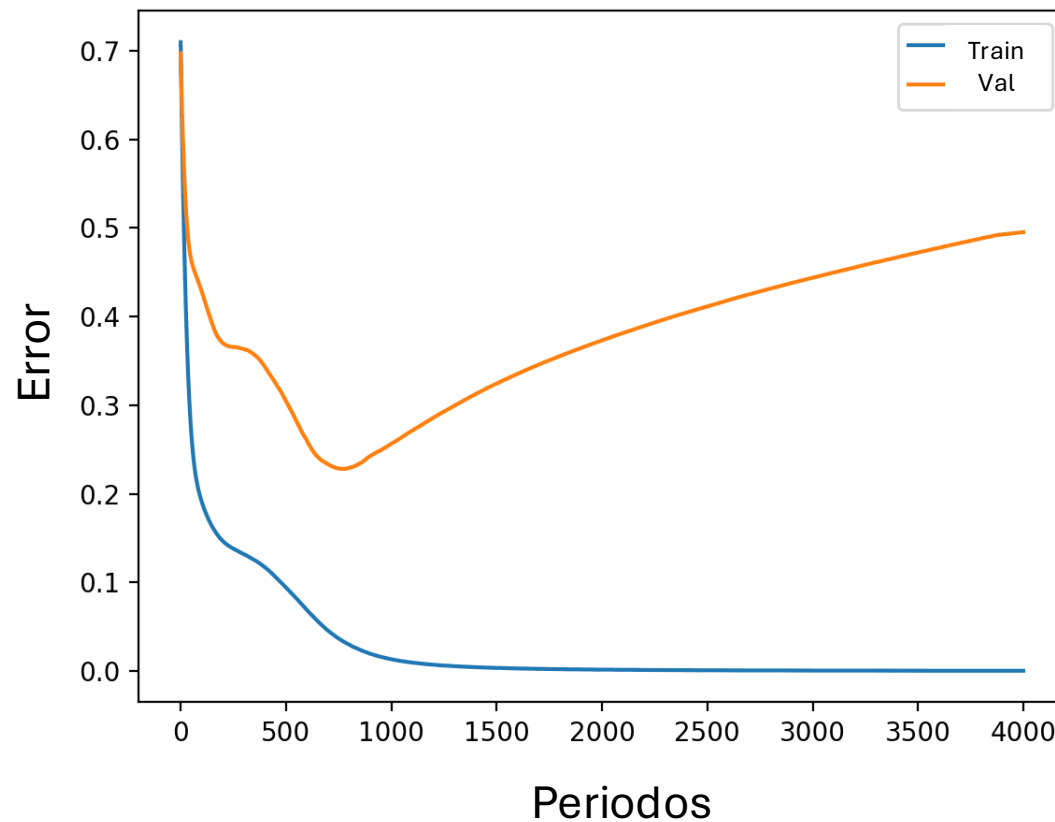
Cuál de estas es sobre ajuste y cuál es falta de ajuste?



Esto es indicador de buen ajuste, sobre ajuste o falta de ajuste?



Esto es indicador de buen ajuste, sobre ajuste o falta de ajuste?



Qué es en cada sector? Sobreajuste, buen ajuste o falta de ajuste?

